Graph Alignment Networks with Node Matching Scores

Evgeniy Faerman[1] **Otto Voggenreiter** [2] **Felix Borutta**[1]
Tobias Emrich[2] **Max Berrendorf**[1] **Matthias Schubert**[1]
[1] *Ludwig-Maximilians-Universität München* Munich, Germany
{faerman, borutta, berrendorf, schubert}@dbs.ifi.lmu.de
[2] Harman International, Germany
{otto.voggenreiter, tobias.emrich}@harman.com

Abstract

In this work we address the problem of graph node alignment at the example of Map Fusion (MF). Given two partly overlapping road networks, the goal is to match nodes that represent the same locations in both networks. For this task we propose a new model based on Graph Neural Networks (GNN). Existing GNN approaches, which have recently been successfully applied on various tasks for graph based data, show poor performance for the MF task. We hypothesize that this is mainly caused by graph regions from the non-overlapping areas, as information from those areas negatively affect the learned node representations. Therefore, our model has an additional inductive bias and learns to ignore effects of nodes that do not have a matching in the other graph. Our new model can easily be extended to other graph alignment problems, e.g., for calculating graph similarities, or for the alignment of entities in knowledge graphs, as well.

1 Introduction

In recent years the high relevance of graph-structured data has generally been accompanied by an increased demand for algorithms that are able to take advantage from the rich body of relational information encoded in graphs. However, in many domains, the entire knowledge base of a certain domain is spread across multiple data sources, e.g., geo-spatial information are spread across various map providing services, social information are spread across multiple social networks, or, somewhat more general, knowledge bases are distributed across various knowledge graph databases.

In this work, we tackle the problem of knowledge fusion by developing a graph neural network model that is able to fuse graph-structured data by learning node matchings. Given two partly overlapping graphs, our approach aims at identifying nodes from both graphs that match to each other. The ultimate goal is to align the graphs such that the information contained in both graphs can be fused properly. In general, the node matching problem is of high relevance in many applications including the fusion of road networks, also called map fusion [1], the matching of knowledge graphs [2, 3, 4, 5, 6, 7, 8, 9, 10], the alignment of social networks [11], or to enable efficient graph comparison [12, 13]. Traditional methods addressing the graph alignment problem typically rely on calculating distances between manually engineered node and edge features. The distances are used to generate pairs of nodes that serve as matching candidates and the set of candidates is subsequently optimized by incorporating the nodes' neighborhood information [1, 14]. However, those approaches require non-trivial, manual parameter tuning, do not generalize well to unseen data and suffer from high complexities which makes them impractical. More recent works have applied graph neural networks (GNNs) which have proven to be particularly suitable for the graph alignment task [8, 9, 10, 12, 13]. The common approach is to aggregate nodes from the local neighborhood into a target node's representation and subsequently compare the resulting representations with each other. More advanced approaches even additionally aggregate nodes from the other graph into the node representations [12, 9]. Either way,

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

the aggregation of information, i.e., the kind of information as well as the locality of the area from which the information is aggregated, is the critical point to get useful embeddings. This holds for the graph alignment task in particular, as information gathered from local node neighborhoods generally tends to become less useful the more the two graphs that shall be aligned differ in terms of structural properties. To overcome the issue of aggregating irrelevant or even misleading information, state-of-the-art approaches use different types of attention mechanisms when aggregating node neighborhoods. The general idea behind using attention is to determine the importance of a certain entity (e.g., a node in the neighborhood) based on an object's own representation (e.g., the representation of the target node) and the current representation of the entity. The higher the importance, the more influence should the corresponding entity have on the object's own representation. However, the attention based on the structure of the own graph is of limited usefulness for the graph alignment task. This problem has recently been addressed, e.g., by additionally aggregating information from the whole counterpart graph into the representation of each node [12]. We argue that the most important information about a node's neighbors is whether they have good alignments in the counterpart graph. Therefore, the Graph Alignment Network (GrAN) presented in this work uses an importance mechanism that, in contrast to previous works, aims at putting special emphasis on nodes that have a good match in the counterpart graph. By doing this, our model effectively introduces an additional inductive bias, i.e., the assumption that neighboring nodes which are likely to be part of the overlapping area of the two graphs are particularly useful for a target node's representation.

After presenting our GrAN model in Section 2, we present preliminary results of our evaluation in Section 3. Precisely, we compare our model against state-of-the-art GNN approaches on map fusion tasks. These tasks turned out to be particularly challenging due to presence of geo-spatial coordinates as they form a natural and very strong baseline. However, it is noteworthy that our model can also be applied to other tasks including knowledge graph matching and the determination of graph similarities.

2 Graph Alignment Networks with Node Matching Scores

Let $G = (V, E, X, P)$ and $G' = (V', E', X', P')$ denote two graphs with V, V' denoting the sets of nodes, E, E' being the sets of edges and X, X', and P, P' being the node and edge attributes, respectively. For the sake of simplicity we assume that both graphs are undirected. Given this setup, we aim at learning a function $F : G \times G' \rightarrow H \in \mathbb{R}^{|V| \times h}$ which takes two graphs as input, applies multiple message passing operations on them and finally retrieves latent vector representations for the nodes of the first graph that was fed as input into F. For the GrAN model, we use a Siamese architecture that allows to apply the same function F to both graphs such that the model's final output are two node embedding matrices $H = F(G, G')$ and $H' = F(G', G)$, respectively. Given the node embeddings, we subsequently align two nodes if their vector representations are considered similar with respect to some similarity function sim and a predefined similarity threshold τ, i.e.,

$$\hat{y}(v_i, v'_j) = \begin{cases} 1 & if \quad sim(h_i, h_j) > \tau \\ 0 & else. \end{cases} , \tag{1}$$

with $v_i \in V$, $v'_j \in V'$, h_i denoting the row vector from H that corresponds to the embedding of node v_i and h_j being the row vector from H' that corresponds to the embedding vector of node v'_j. $\hat{y}(v_i, v'_j) = 1$ indicates that the nodes v_i and v'_j are aligned.

Graph Alignment Network. Graph Neural Networks [15, 16] compute node representations by propagating information between vertices and aggregating them iteratively. In each layer, a message is formed from each source node and passed to its neighbors. Incoming messages for each target node are aggregated and flow into the target node's output representation. Stacking multiple propagation layers allows propagating information over multiple hops. Given a graph $G = (V, E, X, P)$ we define a single message passing propagation step as follows:

$$m_{j \rightarrow i} = M^l(h^l_j, p_{ji}),\ with\ p_{ji} \in P, \tag{2}$$

$$M^l(h^l_j, e_{j,i}) = f_{message}(h^l_j) + LSTM(p_{ji}), \tag{3}$$

$$h^{l+1}_i = [f_{node}(h^l_i), \sum_{j,(j,i) \in E} \alpha^l_{j \rightarrow i} m_{j \rightarrow i}] \tag{4}$$

where, l stands for the current layer, M is a message function, f_{node} and $f_{message}$ are small neural networks and $[\cdot]$ denotes the concatenation operation. Also, note that for the Map Fusion task, the

edge features $p_{ji} \in P$ are sequences of coordinates that represent the road segment. Therefore, we process edge features with a recurrent LSTM network and add the resulting vector to the transformed node representations. However, the inclusion of edge features is generally optional. The weight $\alpha^l_{j\to i}$ determines the importance of the message emitted by node j. In fact, the α weights realize the additional inductive bias of our model, i.e., that the effect that a message has on other nodes is determined by the maximal matching similarity of the source node to the other nodes in other graph. Intuitively, assume that node $v \in V$ has two neighbors $u, w \in V$ with u being aligned to node $u' \in V'$ while node w has no matching node in V'. In this case, our goal is to put high emphasis on u and low emphasis on w when aggregating incoming messages of v. The intuition is that a node $v' \in V'$ who has u' as a neighbor also receives highly weighted information from u' which is assumed to be similar to the information from u. Therefore, in each layer, our approach first determines the best matching for each node with respect to the similarity function sim, and all outgoing messages are weighted accordingly. More formally, we define the message weight $\alpha^l_{j\to i}$ as follows:

$$\alpha^l_{j\to i} = \frac{exp(I(h^l_j, H'^l))}{\sum_{\hat{j}} exp(I(h^l_{\hat{j}}, H'^l))},\ with\ (\hat{j}, i) \in E,\ and \tag{5}$$

$$I(h^l_j, H'^l) = \max_k sim'(f_{match}(h^l_j), f_{match}(h'^l_k)),\ with\ h'^l_k \in H'^l. \tag{6}$$

Since it is not possible to backpropagate through the max operation, we apply the gumbel softmax trick [17, 18]. For the functions $f_{message}$, f_{node} and f_{match}, we use Multilayer Perceptrons and add an additional inductive bias by sharing parameters between these three functions. The inverse Euclidean function is used as similarity function $sim'(a, b) = \frac{1}{d_{euclidian}(a,b)+1}$.

Learning. We use the contrastive loss [19] with positive ϵ^+ and negative ϵ^- margins, i.e.,

$$L = y_{i,j} \cdot max(0, d(h_i, h_j) - \epsilon^+) + (1 - y_{i,j}) \cdot max(0, \epsilon^- - d(h_i, h_j)), \tag{7}$$

to train our model. $y_{i,j}$ is 1 for aligned and 0 for non-aligned pairs of nodes, and d is the Euclidean distance. In contrast to, e.g., triplet loss, the contrastive loss allows us to incorporate nodes having no matchings into the training.

3 Experiments

We evaluate our approach by interpreting the node matching problem as a binary classification task and compare the proposed GrAN model against several state-of-the-art methods including GCN [20], GAT [21], and GraphSAGE [22]. All GNN models use the same siamese architecture with shared weights. For the competitors, as well as for our model we use implementations from the pytorch geometric framework [23]. We also evaluate against a baseline method where we simply use the geo-spatial coordinates of the used road networks' vertices as 2-dimensional node representations for the evaluation. Note, that this is a fairly strong baseline as the graph vertices lie within a continuous space and two matching nodes obviously tend to have similar geo-spatial coordinates. Additionally, where it is possible, we report the results when using probabilistic relaxation [14]. Note, that the probabilistic relaxation method considers all assignments at once using hungarian algorithm [24] in combination with Otsu's [25] method, and therefore can benefit from the fact that graphs are different in size. However, the complexity of $O(c^3)$, with c being the number of matching candidates, is rather high and hence makes it impractical for larger networks. In contrast, all GNN models classify each candidate pair independently. To achieve a fair comparison, we additionally report results when conducting nearest neighbor queries on the set of nodes from the larger graph and query objects only being taken from the smaller graph. This way we still need to find a threshold to divide the retrieved candidates into nodes with and without matchings, but consider the fact that graphs differ in size. Note that this approach ignores cases where nodes match to multiple nodes in counterpart graph.

We train each of the models by using two different road networks (one is from OSM[1] and the other from a commercial map provider), that both cover freeways of the Northrhine-Westphalia region in Germany. In total, the networks consist of 15,442 vertices and 20,166 edges with 2956 of the vertices having a matching node in the other network, i.e., they form matching pairs. In general, nodes can have zero (1:0), one (1:1) or multiple (1:n) matching nodes in the other graph. Every node represents an intersection and each edge corresponds to a road segment represented by the sequence

[1] https://www.openstreetmap.org

of coordinates. To train the models, the matching pairs are split into training, validation and test set (80-10-10). For the purpose of generating negative samples, we project each of the positive sample vertices into the other road network and perform spatial range queries with a radius of approximately 1km^2. The resulting set of vertices form the negative samples of the corresponding node (except for the actual matching vertex). Additionally, we perform the same range queries for the nodes labeled as 1:0 matchings and add the resulting pairs to the negative samples. Finally we get a set of 65,432 positive and negative candidate pairs, that form the final training, validation and test sets as reported in Table 3 in supplementary material. Furthermore, we evaluate how well the models generalize to yet unseen road networks by measuring their matching performance on a pair of road network sections that are taken from the urban area of the city of Munich, Germany. Beside freeways, these graphs also contain types of roads that are not present in the training data, e.g., residential street. The dataset is referred to as MUC. For all models we trained various different architectures whose details and hyperparameter settings can be found in the supplementary material. In the following, we report the results for the architectures that showed the best performance.

Table 1: Resulting F1 Scores for the NRW and MUC datasets.

Method	**NRW**	**MUC**
GrAN	.676	.642
GCN	.091	.009
GAT	.091	.009
GraphSAGE	.091	.009
Spatial Coordinates	.681	.626
Probabilistic Relaxation	.752	-

Table 2: Resulting F1 Scores for the NRW and MUC datasets when using NN queries.

Method	**NRW**	**MUC**
GrAN	.847	.704
GCN	.048	.001
GAT	.071	.005
GraphSAGE	.064	.001
Spatial Coordinates	.827	.661
Probabilistic Relaxation	.752	-

Table 1 shows the F1 scores comparing the classification results of our approach against the results produced by the other GNN based models and when using only spatial coordinates (SC), or the probabilistic relaxation method. While our approach outperforms GCN, GAT and GraphSage, and also the SC baseline on the MUC dataset, we achieve similar matching results as the baseline on the NRW data. When comparing against probabilistic relaxation, the latter achieves better results on the NRW dataset. However, considering the nearest neighbors evaluation (cf. Table 2), the probabilistic relaxation approach is outperformed by both our approach and SC, although all methods except for probabilistic relaxation ignore 1:n matchings. In summary, our method still achieves best results on both datasets.

4 Conclusion

In this work we studied the problem of graph alignment. We presented our ongoing work on a new Graph Neural Network based model, that aggregates information from neighbors based on their alignment scores. We evaluated the proposed approach on Map Fusion tasks and compared it with state-of-the-art models. Our promising experimental results show that the proposed approach outperforms other GNN models by large margins. In future work we plan to adapt our model for other tasks like the alignment of entities in heterogeneous graphs, and also for calculating graph similarities. We further see the potential of improving the runtime of our algorithm by combining it with graph coarsening methods for instance.

Acknowledgements

This work has partially been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A, and has been developed in cooperation with Harman International. The authors of this work take full responsibilities for its content.

[2]Note that some matching nodes representing the same acceleration lanes or freeway exits are indeed located that far from each other.

References

[1] Juan J Ruiz, F Javier Ariza, Manuel A Urena, and Elidia B Blázquez. Digital map conflation: a review of the process and a proposal for classification. *International Journal of Geographical Information Science*, 25(9):1439–1466, 2011.

[2] Zequn Sun, Wei Hu, Qingheng Zhang, and Yuzhong Qu. Bootstrapping entity alignment with knowledge graph embedding. In *IJCAI*, pages 4396–4402, 2018.

[3] Shichao Pei, Lu Yu, Robert Hoehndorf, and Xiangliang Zhang. Semi-supervised entity alignment via knowledge graph embedding with awareness of degree difference. In *The World Wide Web Conference*, pages 3130–3136. ACM, 2019.

[4] Qingheng Zhang, Zequn Sun, Wei Hu, Muhao Chen, Lingbing Guo, and Yuzhong Qu. Multi-view knowledge graph embedding for entity alignment. *arXiv preprint arXiv:1906.02390*, 2019.

[5] Lingbing Guo, Zequn Sun, and Wei Hu. Learning to exploit long-term relational dependencies in knowledge graphs. *arXiv preprint arXiv:1905.04914*, 2019.

[6] Bayu Distiawan Trisedya, Jianzhong Qi, and Rui Zhang. Entity alignment between knowledge graphs using attribute embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 297–304, 2019.

[7] Qiannan Zhu, Xiaofei Zhou, Jia Wu, Jianlong Tan, and Li Guo. Neighborhood-aware attentional representation for multilingual knowledge graphs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 1943–1949. International Joint Conferences on Artificial Intelligence Organization, 2019.

[8] Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. Cross-lingual knowledge graph alignment via graph convolutional networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 349–357, 2018.

[9] Kun Xu, Liwei Wang, Mo Yu, Yansong Feng, Yan Song, Zhiguo Wang, and Dong Yu. Cross-lingual knowledge graph alignment via graph matching neural network. *arXiv preprint arXiv:1905.11605*, 2019.

[10] Yixin Cao, Zhiyuan Liu, Chengjiang Li, Juanzi Li, and Tat-Seng Chua. Multi-channel graph neural network for entity alignment. *arXiv preprint arXiv:1908.09898*, 2019.

[11] Li Liu, William K Cheung, Xin Li, and Lejian Liao. Aligning users across social networks using network embedding. In *IJCAI*, pages 1774–1780, 2016.

[12] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. *arXiv preprint arXiv:1904.12787*, 2019.

[13] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 384–392. ACM, 2019.

[14] Bisheng Yang, Yunfei Zhang, and Xuechen Luan. A probabilistic relaxation approach for matching road networks. *International Journal of Geographical Information Science*, 27(2):319–338, 2013.

[15] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[16] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[17] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[18] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[19] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[20] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[21] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[22] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[23] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[24] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[25] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[26] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

5 Supplementary material

5.1 Dataset statistics

Table 3: Training and test dataset statistics.

	NRW (train)	NRW (test)	MUC (test)
Nodes in G_1	2,430	308	2,996
Nodes in G_2	9,924	1,206	3,153
Candidates	52,138	6,148	378,948
1:1 & 1:n matchings	2,359	294	1,777
1:0 matchings	99	3	1,257

5.2 Architectures & Hyperparameter Settings

For all Graph Neural Network models we tried different architectures and hyperparameters:

Weight Sharing in Siamese Architecture We also tried to learn different encoders for each map without weight sharing. This could help if two maps have different biases. However, it only worsened the results.

Unsupervised Pretraining We used an adapted version of Deep Graph Infomax [26] for pretraining. To corrupt a graph we shifted nodes randomly in coordinate space in each iteration. Model pretraining led to the fastest convergence but did not affect a final result significantly.

Loss We experimented with different positive and negative margins. We obtained best results with a positive margin of 0 and a negative margin of 10. To account for the class imbalance, we decreased the weights for the negative examples inversely proportional to the relative frequency.

Model Depth We tried 1 to 8 message passing layers, followed by 0 to 2 fully connected layers. For GAT we additionally tried 1 to 7 attention heads in each layer. The number of attention heads was constant for all layers.

Width We tested various layer sizes, with both constant and variable sizes of layers in the same model.

Edge Attributes For the fair comparison we adapted the message function of all GNN models to be able to consider edge attributes and we evaluated them with and without edge attributes. We also trained models which consider edge attributes only in the first layer. Using the Douglas–Peucker algorithm we reduced edge attribute sequence to different lengths. We tried the sum and concatenation operations for combining edge and node embeddings in the function M.

Early Stopping We trained all models with early stopping using a patience of 1000.

Message Aggregation To aggregate messages, we tried mean/max pooling and concatenation.

Normalization We normalized the input coordinates to [-1,1] range. Therefore, the data are first zero-centered for each dimension. Next, each dimension is divided by the max distance from the center in all dimensions. This normalization stabilizes training and inference while preserving distance ratios.

Optimizer Settings All tested models were trained with the Adam SGD optimizer with learning rate $lr \in [1\text{e-}4, 1\text{e-}2)$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and the weight decay being chosen from the interval $[0, 1\text{e-}4)$