Bachelor Thesis: Finding a good set of anchors for NodePiece

Renske Diependaal
Supervisor: Michael Cochez
Vrije Universiteit van Amsterdam

July 2022

Abstract

This paper is based on the NodePiece paper written by Galkin et. al. (2022) and builds on it. The goal of NodePiece is to save computational power and storage space. This is done by creating a fixed-size vocabulary, consisting of anchor nodes and relations. The goal of this paper is to find a good strategy to select anchor nodes, which performs better than picking random anchors. The tried strategy uses Personalized PageRank and clustering. For each node, the Personalized PageRank score from that node to all nodes is stored. This gives a co-occurrence matrix, in which each i,jth entry represents the PPR score from node i to node j. This is done using the KGloVe implementation from Cochez et al. (Cochez, Ristoski, Ponzetto, & Paulheim, 2017). The rows from this matrix are then grouped together in n clusters. This is done using k-means clustering and agglomerative clustering. Once the clusters are formed, an anchor node is picked from each cluster, producing n anchor nodes. This is then compared with the original anchor selection strategies from NodePiece, based on: degree, PageRank and random. As an additional test to see if this strategy works, a counter-intuitive strategy is implemented as well. Instead of picking an anchor from each cluster, anchors are picked from the least amount of clusters possible. To test performance, a link prediction task is done. The results suggest that the newly introduced strategy might only make a small difference in finding good anchor nodes.

1 Introduction

A knowledge graph (KG) is a graph consisting of several nodes and edges. The nodes are entities, often relating to instances in the real world. The edges depict relations between the several nodes. A KG stores data in a structured way and this data can be used in machine learning. Common tasks include link prediction, entity resolution and link-based clustering(Nickel, Murphy, Tresp, & Gabrilovich, 2016). To train on these knowledge graphs, a parametrization of all nodes and edges is often necessary (Galkin, Denis, Wu, & Hamilton, 2022).

To do so, all nodes and edges are mapped to a d-dimensional vector. In large graphs, the storage of all parametrizations can take up a lot of space.

To save space, NodePiece takes a different approach (Galkin, Denis, et al., 2022). Instead of encoding all nodes and edges with a d-dimensional vector, it creates a fixed-size vocabulary. The vocabulary V consists of anchor nodes A and all relation types R ($V = A + R$), where the number of anchor nodes $|A|$ is much smaller than the number of nodes $|N|$ in the graph. This gives $|V| \ll |N|$. Each node can be encoded using the k-nearest anchors and the relation types. This reduces the number of parameters needed.

Anchor nodes are nodes selected from the KG. In their paper, Galkin et. al. try three different strategies to select good anchor nodes. The first strategy is based on the degree of the nodes, one strategy based on PageRank and the last strategy is to select nodes randomly. Testing these strategies with a link prediction experiment shows that PageRank and degree perform only slightly better than random.

The goal of this paper is to find a strategy to select anchor nodes that performs better than random selection. The strategy that was tried is based on Personalized PageRank (PPR). Using the KGloVe implementation, a sparse rank vector is created for each node in the knowledge graph. These vectors form the rows of a co-occurrence matrix. The rows are grouped together in a number of clusters, from which the anchor nodes are selected. To test performance of the anchor nodes, a (transductive) link prediction experiment is conducted. Two knowledge graphs are used, FB15k-237 and WN18RR (Toutanova & Chen, 2015).

2 NodePiece

NodePiece has three different strategies to pick n anchor nodes. The first one is based on the degree of each node. All graphs have inverse edges added, so there is no difference between the in-degree and out-degree. This strategy picks the n nodes with the highest degree as anchor nodes. The second strategy looks at PageRank. The PageRank for the graph is calculated, giving every node a PageRank score. The n highest scoring nodes are picked as anchors. The third strategy is to randomly pick n nodes.

Once the anchor nodes are selected, they can be used to tokenize nodes. Each node is tokenized using the k nearest anchors, the distance from the node to these anchors and the outgoing edges related to the node. A visualization of this is shown in Fig 1. The k nearest anchors are found using Breadth First Search. The distance from the node to the anchors and the added relations help to maintain a sense of the underlying graph structure.

After tokenization, NodePiece trains for the link prediction task using a stochastic local closed world assumption training approach, a SLCWA training loop from the pykeen library. The training loop makes use of the NodePiece-Rotate model. Just as in the original RotatE model (Sun, Deng, Nie, & Tang, 2019), NodePiece-Rotate makes use of the Hadamard product between the head

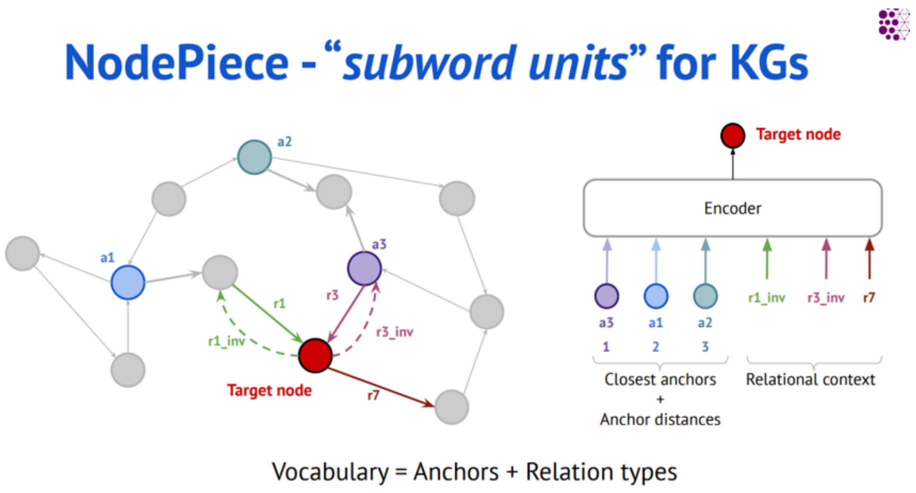

NodePiece - "subword units" for KGs
a1
a2
a3
r1
r1_inv
r3
r3_inv
r7
Target node
Encoder
Target node
a3 1
a1 2
a2 3
r1_inv
r3_inv
r7
Closest anchors + Anchor distances
Relational context
Vocabulary = Anchors + Relation types

Figure 1: Illustration of the tokenization of nodes in NodePiece.

and the relation in the complex space. Using the Hadamard product allows the model to identify symmetric/antisymmetric, inversion, and composition patterns. The embedding of head h is rotated using the following formula:

$$h_{rot} = [h_{re} * r_{re} - h_{im} * r_{im},\ h_{re} * r_{im} + h_{im} * r_{re}]$$

Where h is the embedding of the head of a triple and r the embedding of the relation. The first part corresponds to the real part of the rotated head, and the second part to the imaginary part. The scoring function is then given by $h_{rot} - t$, where t is the embedding of the tail. The closer this score is to zero, the better.

The embedding of the head, relation and tail is created by passing the tokenization through a two-layered multilayer perceptron. It consists of a Linear layer, followed by a DropOut layer, an activation ReLu layer and then another Linear layer.

In the NodePiece paper, several experiments are conducted: transductive link prediction, inductive link prediction, out-of-sample link prediction, node classification and relation prediction. This paper focuses on transductive link prediction. Performance in a link prediction task says something about the ability of being able to predict a link between nodes. This task tries to predict the head and tail (separately) based on the triple $< head, relation, tail >$. Either head or tail is unknown.

For transductive link prediction, training is done on part of the graph and testing on a disjoint part of the same graph. For inductive link prediction, training is done on a graph and testing is done on part of a new (disjoint) inference graph with entities unseen in training. Fig 2 shows a visualization of this.

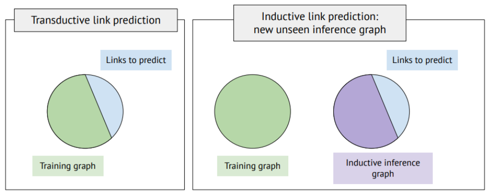

Transductive link prediction
Links to predict
Training graph
Inductive link prediction: new unseen inference graph
Links to predict
Training graph
Inductive inference graph

Figure 2: Illustration of the difference between transductive- and inductive link prediction (Galkin, Berrendorf, & Hoyt, 2022).

For out-of-sample link prediction, the validation and test set contain entities that are not in the training set. These entities do have a few edges connected to nodes in the training set. Relation prediction is similar to link prediction, in that they both try to complete a triple $< head, relation, tail >$. Relation prediction tries to predict the relation in the triple, given a head and tail. Node classification tries to predict the value of a node, based on the labels of neighbouring nodes.

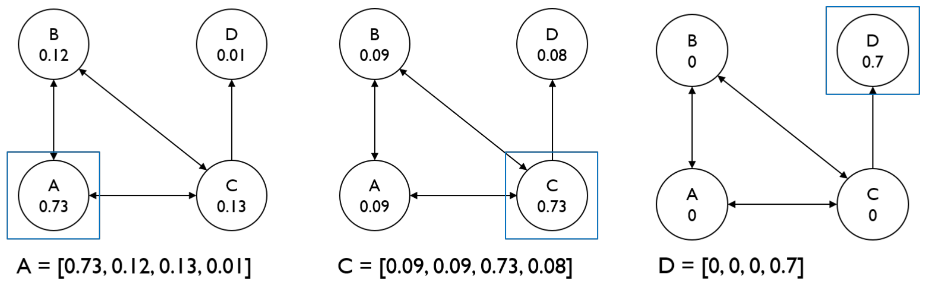
To perform well in the link prediction experiment, the model should be able to tell the nodes and relations apart. To be able to distinguish between all nodes, the tokenization should be as unique as possible. Not focusing on the relational context, this means that the sequence of k nearest anchors and their distances should be as unique as possible for each node. This paper aims to find such anchor nodes using Personalized PageRank as an anchor selection strategy.

3 Personalized PageRank

Personalized PageRank is a variant of PageRank (Page, Brin, Motwani, & Winograd, 1999). The PageRank score of a node is a measure for the centrality of the node. The more incoming edges a node has, the higher the rank. Rank for a node is also increased if the nodes with a relation to the node have a high rank. The PageRank paper explains that the simple version of PageRank can be seen as the probability distribution of a random surfer on the internet. The web pages correspond to nodes in the graph, links to the edges. The surfer starts at some node and moves to the next node using a random out-going edge, then to the next node, etc. The random walk takes as many steps as needed, until the difference in probability distribution between two steps changes less than some value β.

The simple version of PageRank can get stuck at a node with no out-going edges. To prevent this, a random jump was introduced. The surfer can make a jump to any page, with probability α. In this paper, $\alpha = 0.3$. The jump is given by the probability distribution v over all nodes. In PageRank, this distribution is uniform. The jump also prevents the walk from getting stuck in so-called spider traps (Leskovec, Rajaraman, & Ullman, 2020). A spider trap is a group of nodes that have links between them, but no out-going edges to other nodes.

Personalized PageRank is similar to PageRank, the difference is in the way the walk jumps to a random node. In PageRank, the walk can jump to any node with uniform probability. In Personalized PageRank, these probabilities can be changed. In this paper, all jumps teleport back to the starting node. This means that each starting node produces a different probability distribution, see Fig 3 for an example. In the figure, the probability distributions are shown below the graph in vector form.



(a) Start node A (b) Start node C (c) Start node D

Figure 3: Example of the difference in probability distribution in different starting nodes. On the left, node A is the start node. In the middle, node C and on the right node D as indicated by the blue squares. Below the graph, the probability distribution is shown in form of a vector.

This paper makes use of the Fast All-Pairs PPR Algorithm introduced in the Global RDF Vector Space Embeddings paper (Cochez et al., 2017). This algorithm is based on the Bookmark-Coloring Algorithm (Berkhin, 2006). BCA gives an approximation of PPR, but only for the nodes that will receive a significant rank. The algorithm is described as distributing a unit amount of paint to the start node. This paint is partly retained by the node and the other part is distributed over the out-going edges. This process repeats for the nodes attached to those out-going edges. At one point the amount of paint being distributed is smaller than $\epsilon = 0.00001$, then the algorithm stops and any nodes without paint get rank 0. This creates a sparse rank vector.

Cochez et al. make the BCA algorithm more efficient by changing the order of nodes for which the rank vector is calculated. The rank vector (BCV) for a node is dependent on the BCV of the nodes that are reachable with one hop. These BCVs are calculated before the BCV of the node that is dependent on them. This allows the algorithm to re-use calculations of BCV, instead of calculating some BCVs multiple times.

Using this algorithm, the PPR vector is calculated for each node in the graph, using that node as starting node. These vectors together form the rows of a PPR matrix. The same is done for the graph with reversed edges, giving a PPR matrix as well. Using the KGloVe algorithm, these two matrices are then summed and together form a co-occurrence matrix (Cochez et al., 2017).

Since the vectors used to form this matrix are sparse, this matrix is sparse as well. This leads to better scalability. Following the example from Figure 3, the co-occurrence matrix is shown in Figure 4.

$$\begin{array}{c|cccc|} & \text{A} & \text{B} & \text{C} & \text{D} \\ \text{A} & 0.73 & 0.12 & 0.13 & 0.01 \\ \text{B} & 0.12 & 0.73 & 0.13 & 0.01 \\ \text{C} & 0.09 & 0.09 & 0.73 & 0.07 \\ \text{D} & 0 & 0 & 0 & 0.7 \end{array}$$

Figure 4: Example illustration of a sparse matrix containing PPR scores, for nodes A. B, C and D. Each row corresponds to the rank vector calculated using the corresponding node as start node.

PPR is a way of measuring node-to-node proximity (Page et al., 1999). If two nodes have a similar PPR vector, they are likely to be able to reach the same the nodes in the same way. Intuitively, picking similar nodes as anchor nodes would give a tokenization of nodes that is also similar and it would be hard to distinguish between nodes. To prevent this, the next step in the anchor selection is to group the vectors in the matrix together in a number of clusters, using a distance metric. That is, the rows are grouped.

4 Clustering

Data clustering algorithms combine data points into clusters, such that the points in a cluster are similar to each other according to some metric (Madhulatha, 2012). In this case, all nodes are clustered, based on the corresponding PPR vectors. In this paper, the number of clusters is the same as the number of anchors wanted. According to Madhulatha(2012), clustering algorithms are either hierarchical or partitional. Partitional algorithms build all clusters at the same time. Hierarchical algorithms build clusters by merging or splitting them successively (Pedregosa et al., 2011). In this paper, two clustering algorithms were used. The first one is k-means clustering, which is a partitional algorithm. The second is agglomerative clustering, which is a bottom-up hierarchical algorithm

4.1 K-means

K-means clustering requires a number n of clusters as input (Pedregosa et al., 2011). The algorithm initially picks n so-called centroids, distant from each other. Then each sample in the data is assigned to the nearest centroid. This gives n clusters, which then get assigned a new centroid based on the mean of the cluster. Samples are again assigned to the nearest centroid and the clusters are updated until the centroids change less than a given threshold. The aim is to minimize the inertia of the clusters, given by:

$$\sum_{i=0}^{n} \min_{\mu_j \in C}(||x_i - \mu_j||^2)$$

Where n is the number of samples, x is a sample and C is the set of all clusters, each with a mean μ.

4.2 Agglomerative clustering

Agglomerative clustering starts of with all samples as individual clusters (Pedregosa et al., 2011). Then, at each step, the two closest clusters are merged together. This process repeats until the desired number of clusters has been reached. In this paper, the single linkage metric was used. This means that the two closest clusters get determined by comparing the smallest distance between pairs of clusters, and this distance is minimized. The smallest distance is calculated using Euclidean distance.

5 Anchor selection strategy

After the clusters have been created, the anchors are picked. This is done in two strategies for this project: Spread and Focused.

Spread picks anchor nodes spread over all the clusters as much as possible. Given n clusters, it will pick a random node from cluster 1, cluster 2, ..., cluster n. Focused does the opposite and tries to pick as much anchor nodes from one cluster as possible. It picks the largest cluster and focuses on this cluster, selecting all nodes in this cluster. If the cluster has less nodes than anchors needed, Focus moves on to the second largest cluster and picks nodes from this cluster, until enough anchors have been selected.

The goal of Spread is to have anchor nodes that differ a lot from each other and this strategy follows the intuition described earlier. The more different the nodes are, the more distinguishable the tokenized nodes are and the better the model should perform. Focused seems counter-intuitive to this logic and is also not designed to perform well. The expectation is that Focused performs worse and this would show that the strategy of picking anchors from these clusters has an impact on performance.

To summarize, the anchor selection strategy consists of first running Personalized PageRank for each node in the graph. This gives a sparse co-occurrence matrix, whose vectors are then clustered using two different techniques: k-means and agglomerative clustering. From these clusters, the anchor nodes are selected in two different ways as well: Focused and Spread. A sketch of the complete anchor selection strategy is shown in Fig 5. This figure also shows the different ways in which clustering and anchor selection are combined.

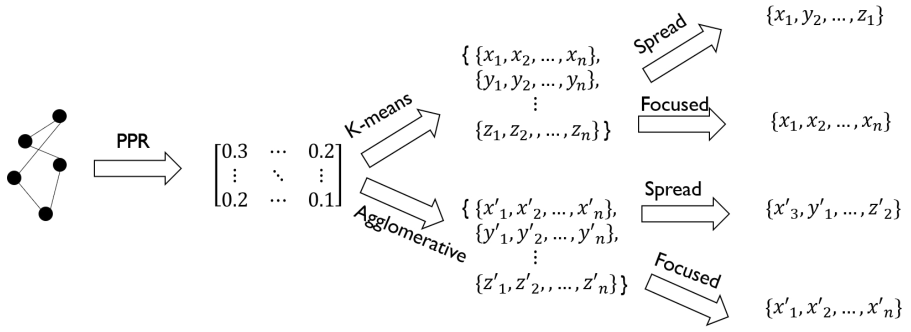


Figure 5: Anchor Selection strategy. Starting with a knowledge graph, PPR is applied to all nodes in the knowledge graph, giving a matrix. The vectors in the matrix are clustered using k-means and agglomerative clustering separately. From the clusters, anchor nodes are selected, either by Focused or Spread.

6 Link prediction experiment

After a selection of anchor nodes has been made, their performance is tested using a link prediction experiment. This experiment consists of two parts, head prediction and tail prediction (Berrendorf, Faerman, Vermue, & Tresp, 2020). For each triple $< head?, relation, tail >$ in the test set, head prediction tries to predict which entity is the head. All entities get a score for how likely they are the head and are then ranked accordingly. The final rank is the rank of the true head. The same is done in tail prediction, except that the tail is predicted instead of the head. The final rank is given by combining the rank of tail- and head prediction. A realistic rank is used. So if multiple entities have the same prediction score as the true entity, the true entity is assumed to be ranked in the middle.

Several evaluation metrics are used. The first one is mean rank. This is the computed rank, averaged over all triples in the test set. The lower, the better.

The second one is mean reciprocal rank. This is the multiplicative inverse of the mean rank and is given by the following formula:

$$\frac{1}{|R|}\sum_{r\in R}\frac{1}{r}$$

Where R is the set of all mean ranks r. This formula gives a number between 0 and 1. An advantage of this metric is that it is less influenced by outliers (Liu et al., 2019). The closer to one, the better.

The third metric is hits@k, for k = 1 and k = 10. This represents the percentage of triples, for which the true entity was ranked in the top k. For this metric, the place of triples not ranked in the top k does not matter. it does not make a difference if a triple was ranked at 11 or 100 (Berrendorf et al., 2020).

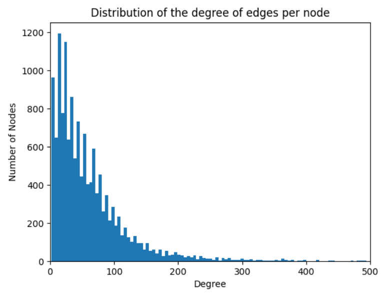
The last metric is the adjusted mean rank. This is defined to be the mean rank, divided by the expected value of the mean rank (Berrendorf et al., 2020). So this metric represents the mean rank, except it is adjusted to the size of the model. This makes it easier to compare different models.

7 Datasets

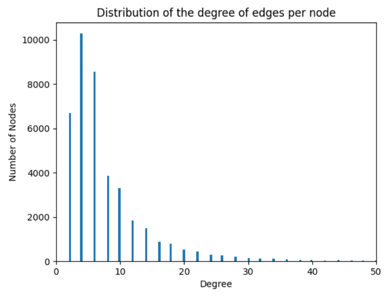
Two datasets are used in this paper. The first one is WN18RR, loaded using the pykeen library. It has a vocabulary size of 40k and an embedding dimension of 1000. It is a sparse graph. It uses 500 anchors and nodes are tokenized using 50 anchors per node.

The second one is FB15k-237, also loaded using pykeen. It has a vocabulary size of 15k and an embedding dimension of 2000. It is a denser graph than WN18RR. It uses 1000 anchors and tokenizes nodes using 20 anchors per node (Galkin, Denis, et al., 2022). This graph uses less anchors per node as WN18RR, because it is more dense. The nodes are more easily connected to the anchor nodes.

In Figure 5, the degree distribution is given for both graphs. FB15k-237 has a few nodes with a high degree, whereas WN18RR has a lot of nodes with a low degree. This also shows the difference in density. The high degree nodes in FB15k-237 make the graph more connected than WN18RR.



(a) FB15k-237



(b) WN18RR

Figure 6: The degree distribution of graphs FB15k-237 and WN18RR. Note the difference in scale on the x-axis.

8 Results

The experiment was run on two pykeen datasets: FB15k-237 and WN18RR. All metrics are calculated for tail prediction, head prediction and both combined. Below are the raw results for both combined. Table 1 contains the results for FB15k-237, Table 2 for WN18RR. The tables also reports the scores for the anchor selection strategies used in the NodePiece paper. For the raw results of head and tail prediction separately, see appendix B.

Table 1: Table containing the results of the link prediction experiment for FB15k-237. Degree, PageRank and random are anchor selection strategies used in NodePiece. A downwards arrow indicates that a lower score is better for this metric, an upwards arrow indicates the higher the score, the better. MRR stands for mean reciprocal rank, AMR for adjusted mean rank.

FB15k-237 both	Mean rank ↓	MRR ↑	H@1 ↑	H@10 ↑	AMR ↓
k-means spread	302.0184	0.2585	0.1747	0.4221	0.0442
k-means focused	310.0953	0.2556	0.1732	0.4161	0.0453
agglomerative spread	303.745	0.2569	0.1741	0.4185	0.0444
agglomerative focused	309.0503	0.2526	0.1679	0.4184	0.0452
degree	307.532	0.2536	0.1697	0.4179	0.0451
pagerank	310.3791	0.2558	0.1697	0.4188	0.0455
random	311.6496	0.2563	0.1734	0.4169	0.0457

Table 2: Table containing the results of the link prediction experiment for WN18RR. Degree, PageRank and random are anchor selection strategies used in NodePiece. A downwards arrow indicates that a lower score is better for this metric, an upwards arrow indicates the higher the score, the better. MRR stands for mean reciprocal rank, AMR for adjusted mean rank.

WN188RR both	Mean rank ↓	MRR ↑	H@1 ↑	H@10 ↑	AMR ↓
k-means spread	1455.0205	0.3931	0.3264	0.5111	0.0718
k-means focused	1445.0489	0.381	0.3104	0.4979	0.0713
agglomerative spread	1421.5518	0.4026	0.3423	0.5058	0.0702
agglomerative focused	1522.1948	0.3541	0.2842	0.4749	0.0751
degree	1475.477	0.3958	0.3309	0.5026	0.0728
pagerank	1490.6216	0.396	0.329	0.5127	0.0736
random	1502.8004	0.4014	0.3391	0.5087	0.0742

Presenting these two tables differently makes it easier to compare the performance of the Spread and Focused strategy. See Table 3.

Comparing the results of the two different clustering techniques in FB15k-237, k-means performs slightly better than agglomerative clustering on all metrics. This could be explained by the fact that (in this setup) agglomerative clustering tries to minimize the minimal distance between two clusters, while k-means focuses on the inertia of the clusters. The inertia is targeted at all vectors in a cluster, making all vectors more similar, instead of finding a pair of clusters for which two vectors happen to have a low distance between them.

Comparing the Spread and Focused strategy, Spread seems to do better on all metrics, for all clustering strategies. The difference is not very large. Intuitively, the two strategies should have the opposite effect of each other. Spread chooses anchors that should not be similar, whereas Focused picks anchors that should be similar. The small difference between them could suggest that the use of PPR and clustering has only a small impact on finding good anchors.

Table 3: Tables containing the results of the link prediction experiment for the newly implemented strategies.

(a) Mean reciprocal rank ↑

MRR ↑	Spread	Focused
Agglo FB	0.2569	0.2526
Agglo WN	0.4026	0.3541
K-means FB	0.2585	0.2556
K-means WN	0.3931	0.381

(b) Adjusted mean rank ↓

AMR ↓	Spread	Focused
Agglo FB	0.0602	0.0618
Agglo WN	0.0702	0.0751
K-means FB	0.0601	0.0611
K-means WN	0.0718	0.0713

(c) hits@1 ↑

H@1 ↑	Spread	Focused
Agglo FB	0.1741	0.1679
Agglo WN	0.3423	0.2842
K-means FB	0.1747	0.1732
K-means WN	0.3264	0.3104

(d) hits@10 ↑

H@10 ↑	Spread	Focused
Agglo FB	0.4185	0.4184
Agglo WN	0.0.5058	0.4749
K-means FB	0.4221	0.4161
K-means WN	0.5111	0.4979

Using Table 1 and 2 to compare the Focused strategy with the random strategy used in NodePiece, focused performs better than random on mean rank, adjusted mean rank and once on H@10. On the other metrics, it performs slightly worse. The expectation was that the Focused strategy would have a negative influence on selecting good anchor nodes, but the differences are actually very small.

The Spread strategy seems to perform slightly better than the strategies used in NodePiece (degree, PageRank and random) on the FB15k-237 KG. On the WN18RR KG, the Spread strategy has a better mean rank and adjusted mean rank, but it performs worse on the mean reciprocal rank and on hits@1. This suggests that Spread has less triples that are very high ranked, but a better mean performance.

It is also worth to note the large difference between head and tail prediction results. See Tables 4 and 5. In the FB15k-237 dataset, tail prediction performs much better than head prediction. Mean rank differs from around 400 to a little above 200. For WN18RR, tail prediction performs better than head prediction as well, but the difference is smaller.

The difference between head and tail prediction is caused by the relational context. The smaller difference between head and tail prediction for WN18RR seems to enforce the idea that relational context plays a larger role in FB15k-237 than it does for WN18RR. This was also found in an ablation study in the NodePiece paper. After leaving out the anchor nodes and only using relational context, performance of FB15k-237 was only slightly impacted, in contrast to WN18RR. The H@10 performance of WN18RR dropped to zero. This could be explained by the fact that FB15k-237 has more unique relations than WN18RR, giving a more unique hashing of nodes.

Table 4: Tables showing the difference between head and tail prediction results for FB15k-237, k-means clustering strategy

FB15k-237		Mean rank ↓	MRR ↑	H@1 ↑	H@10 ↑	AMR ↓
k-means spread	head	399.8853	0.1727	0.1061	0.3069	0.0601
	tail	204.1514	0.3444	0.2432	0.5372	0.0293
k-means focused	head	406.8995	0.1671	0.0996	0.30144	0.0611
	tail	213.2912	0.344	0.2469	0.5307	0.0296

Table 5: Tables showing the difference between head and tail prediction results for WN18RR, k-means clustering strategy

WN188RR		Mean rank ↓	MRR ↑	H@1 ↑	H@10 ↑	AMR ↓
k-means spread	head	1716.4501	0.3897	0.3249	0.5062	0.0847
	tail	1193.591	0.3966	0.328	0.5161	0.0589
k-means focused	head	1716.4754	0.3725	0.3068	0.4818	0.0847
	tail	1173.6224	0.3893	0.314	0.514	0.0579

9 Future work

In this paper, only one of the five tasks of NodePiece was performed. To compare the performance of the strategies more, the newly introduced strategy could also be tested using the other tasks: inductive link prediction, relation prediction, out-of-sample link prediction and node classification.

For future work within the Personalized PageRank and clustering strategy, it could be useful to look at different distance metrics. Instead of using Euclidean distance, Manhattan distance might make more sense. This distance metric looks at the absolute difference between entries of vectors, instead of squared difference (Madhulatha, 2012).

It could also be interesting to look at different ways to create the co-occurrence matrix that was now created using PPR. One way could be to use only the distance to other nodes. This would not give nodes that are dissimilar, but the distances that are eventually used to tokenize nodes should be different. Perhaps just the distance to anchor nodes is enough to be able to distinguish nodes.

10 Conclusion

The anchor selection strategy introduced in this paper was applied to two knowledge graphs, FB15k-237 and WN18RR. After the anchor selection, a link prediction experiment was conducted. The new strategy seems to perform slightly better than the strategies used in the NodePiece paper. The new strategy performs better on all metrics, for the FB15k-237 knowledge graph. For the WN18RR KG, the mean performance of the new strategy is better, but the results suggest that the old strategies might have very high ranked triples more often. The new strategy seems to perform slightly better than random.

The small difference between the Spread and Focused strategy could suggest that Personalized PageRank combined with clustering has only a small effect on picking good anchor nodes.

References

Berkhin, P. (2006). Bookmark-Coloring Algorithm for Personalized PageRank Computing. *Internet Mathematics*, *3*, 41 - 62.

Berrendorf, M., Faerman, E., Vermue, L., & Tresp, V. (2020, February). On the Ambiguity of Rank-Based Evaluation of Entity Alignment or Link Prediction Methods. *arXiv e-prints*, arXiv:2002.06914.

Cochez, M., Ristoski, P., Ponzetto, S. P., & Paulheim, H. (2017). Global RDF vector space embeddings. In C. d'Amato et al. (Eds.), *The semantic web - ISWC 2017 - 16th international semantic web conference, vienna, austria, october 21-25, 2017, proceedings, part I* (Vol. 10587, pp. 190–207). Springer. doi: 10.1007/978-3-319-68288-4_12

Galkin, M., Berrendorf, M., & Hoyt, C. T. (2022). An Open Challenge for Inductive Link Prediction on Knowledge Graphs. *CoRR*, *abs/2203.01520*. doi: 10.48550/arXiv.2203.01520

Galkin, M., Denis, E., Wu, J., & Hamilton, W. L. (2022). NodePiece: Compositional and parameter-efficient representations of large knowledge graphs. In *International conference on learning representations.* Retrieved from `https://openreview.net/forum?id=xMJWUKJnFSw`

Leskovec, J., Rajaraman, A., & Ullman, J. (2020). *Mining of massive datasets* (3rd ed.). Cambridge University Press. Retrieved from `http://mmds.org/`

Liu, Y., Li, H., García-Durán, A., Niepert, M., Oñoro-Rubio, D., & Rosenblum, D. S. (2019). MMKG: multi-modal knowledge graphs. In P. Hitzler et al. (Eds.), *The semantic web - 16th international conference, ESWC 2019, portorož, slovenia, june 2-6, 2019, proceedings* (Vol. 11503, pp. 459–474). Springer. doi: 10.1007/978-3-030-21348-0_30

Madhulatha, T. S. (2012). An overview on clustering methods. *CoRR.* doi: 10.48550/ARXIV.1205.1117

Nickel, M., Murphy, K., Tresp, V., & Gabrilovich, E. (2016). A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, *104*(1), 11-33. doi: 10.1109/JPROC.2015.2483592

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999, November). *The PageRank Citation Ranking: Bringing Order to the Web.* (Technical Report No. 1999-66). Stanford InfoLab. Retrieved from `http://ilpubs.stanford.edu:8090/422/` (Previous number = SIDL-WP-1999-0120)

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Sun, Z., Deng, Z.-H., Nie, J.-Y., & Tang, J. (2019). *RotatE: Knowledge graph embedding by relational rotation in complex space.* arXiv. doi: 10.48550/ARXIV.1902.10197

Toutanova, K., & Chen, D. (2015, July). Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd workshop on continuous vector space models and their compositionality* (pp. 57–66). Beijing, China: Association for Computational Linguistics. Retrieved from `https://aclanthology.org/W15-4007` doi: 10.18653/v1/W15-4007

A Github

Link to the github repository: `https://github.com/RenskeDiep/Bachelor_Thesis`

B Results of the head and tail prediction

The following tables present the results of head prediction and tail prediction separately:

Table 6: Table containing the results of the head prediction experiment for FB15k-237

FB15k-237 head	Mean rank ↓	MRR ↑	H@1 ↑	H@10 ↑	AMR ↓
k-means spread	399.8853	0.1727	0.1061	0.3069	0.0601
k-means focused	406.8995	0.1671	0.0996	0.30144	0.0611
agglomerative spread	400.575	0.1686	0.1011	0.3028	0.0602
agglomerative focused	411.3362	0.1694	0.1027	0.301	0.0618
degree	409.1416	0.1692	0.1018	0.3039	0.0616
pagerank	414.9797	0.1702	0.1003	0.3058	0.0285
random	412.9332	0.169	0.1023	0.3013	0.0623

Table 7: Table containing the results of the tail prediction experiment for FB15k-237

FB15k-237 tail	Mean rank ↓	MRR ↑	H@1 ↑	H@10 ↑	AMR ↓
k-means spread	204.1514	0.3444	0.2432	0.5372	0.0293
k-means focused	213.2912	0.344	0.2469	0.5307	0.0296
agglomerative spread	206.9151	0.3453	0.2472	0.5342	0.0287
agglomerative focused	206.7644	0.3359	0.2329	0.5359	0.0287
degree	205.9232	0.3379	0.2376	0.5319	0.0285
pagerank	205.795	0.3405	0.2392	0.5319	0.0625
random	210.3607	0.3436	0.2445	0.5326	0.0292

Table 8: Table containing the results of the head prediction experiment for WN18RR

WN188RR head	Mean rank ↓	MRR ↑	H@1 ↑	H@10 ↑	AMR ↓
k-means spread	1716.4501	0.3897	0.3249	0.5062	0.0847
k-means focused	1716.4754	0.3725	0.3068	0.4818	0.0847
agglomerative spread	1674.3218	0.4002	0.3410	0.4983	0.0826
agglomerative focused	1804.3003	0.3618	0.2798	0.4600	0.0891
degree	1743.9767	0.3925	0.3297	0.4969	0.0861
pagerank	1797.3793	0.3929	0.328	0.5075	0.0887
random	1792.6053	0.3983	0.3393	0.4997	0.0885

Table 9: Table containing the results of the tail prediction experiment for WN18RR

WN188RR tail	Mean rank ↓	MRR ↑	H@1 ↑	H@10 ↑	AMR ↓
k-means spread	1193.591	0.3966	0.328	0.5161	0.0589
k-means focused	1173.6224	0.3893	0.314	0.514	0.0579
agglomerative spread	1168.7818	0.4049	0.3437	0.5133	0.0577
agglomerative focused	1240.0893	0.3619	0.2886	0.4897	0.0891
degree	1206.9764	0.3992	0.3321	0.5082	0.0596
pagerank	1183.8629	0.3991	0.33	0.5178	0.0585
random	1212.9956	0.4046	0.3389	0.5178	0.0599