Graph Embeddings for Enrichment of Historical Data*

J. Baas[1][0000−0001−8689−8824], M. M. Dastani[1][0000−0002−4641−4087], and A. J. Feelders[1][0000−0003−4525−1949]

Utrecht University, Heidelberglaan 8, 3584 CS Utrecht, Netherlands

Abstract. In this work-in-progress paper we describe our method of combining expert knowledge and RDF graph embeddings to solve for specific downstream tasks such as entity resolution. We show that efficiency gains can be made by choosing the correct gradient descent algorithm and that expert input can lead to the desired results.

Keywords: Graph Embedding · RDF · GloVe · Entity Resolution · Digital Humanities

1 Introduction

The internet has seen the emergence of numerous knowledge bases modeled as *knowledge graphs* of various sizes and structures, collectively called `Linked Open Data`. The combination of multiple RDF data sources and extracting their combined knowledge is the purpose of the *Golden Agents* [3] project. Via a multi-agent system the complexity of using multiple sources to answer a single query is hidden from the user. The long term objective of our work is to provide an *enrichment* to query results by using machine learning on knowledge graphs. Examples of query enrichment can vary from providing suggestions to users while writing queries to linking entities as likely duplicates or adding potentially relevant information to an otherwise empty result. Most data mining and machine learning methods, however, require data to be in propositional form with a vector of values associated with each data point. A knowledge graph, therefore, first has to be converted to this propositional form before these methods can be applied. Thanks to advances in their Natural Language Processing (NLP) counterparts, many techniques adapted from NLP have become available to assist with this particular problem. Many though, are designed for the creation of general purpose embeddings without the inclusion of expert opinion regarding a particular application.

In the context of the Golden Agents project, we are interested in application specific tasks such as entity resolution by using historical expert knowledge in building application specific embeddings. In general, we would like to use off the shelf machine learning algorithms to solve tasks such as entity resolution.

* Supported by the Netherlands Organisation for Scientific Research.

However, since we are dealing with RDF graphs, in which edges have a specific semantic content, we would like to leverage this extra information with domain specific expert knowledge. This confronts us with a set of problems:

1. Which node neighborhood creation strategies work best for certain historical data sets with a specific graph structure.
2. Can the embedding process be made more efficient so embeddings can be created on demand.
3. Can a domain expert feed information to the system as to affect the resulting embedding in a predictable fashion.

In order to address these problems we generate a context (neighborhood) for each node in the RDF graph. The context will take into account the importance of the relations with the neighboring nodes. This importance, which is represented by a weight, is provided by domain experts. We experiment with multiple strategies of creating neighborhoods, exploiting properties of RDF graphs such as the fact that some nodes can only have incoming edges. These neighborhoods are then used as input to embed each node in a d-dimensional space, where d is chosen by the embedding designer, using an algorithm originally designed for embedding words. We also experiment with different stochastic gradient descent algorithms to see which performs best on our problem set.

We show that the neighborhood creation strategy can have a significant impact on how an embedding can be used for an application specific purpose such as entity resolution. We demonstrate that a specific choice of weights can give desired outcomes in the structure of an embedding. In the realm of embedding optimization, we show that choosing the right type of stochastic gradient descent algorithm can give large improvements in convergence time over the type that is usually mentioned in literature. Additionally, a small improvement in the quality of fit can sometimes be obtained.

In this work-in-progress paper we describe in section 2 previous work that has inspired our approach. In section 3 we discuss the historical data that has been made available to us in the Golden Agents project. Then, in section 4, we explain our approach in general terms and how subsequently it differs from the related work. The process of deriving features from an RDF graph is explained in detail in section 5. Section 6 describes how nodes are embedded using the features described in the section 5 and also describes some improvements on the gradient descent optimization process. We explain the details of implementation in section 7. Section 8 shows some of our preliminary results. Finally, in section 9 we give future research directions.

2 Related Work

Since Word2Vec [10] came on the scene, it has inspired many techniques that leverage the same technology to embed objects other than words. Notable examples of this work are Deepwalk [14] and Node2Vec [8]. Both Deepwalk and Node2Vec allow for the creation of derived features for nodes in a graph by using

a random walker to generate a context. In the case of Deepwalk, a uniform random walker is used, while Node2Vec extends this idea and introduces a random walker that can be either biased towards local neighborhoods or more global graph structures. These features are then usually embedded in a space using the Skip-gram algorithm in combination with negative sampling [11]. Additionally, in the sphere of Linked Open Data, RDF2Vec [16] also adapts natural language models to create a set of token sequences (a sentence analogue) using graph walks, embedding them in a similar fashion.

In contrast to [14,8,16], Cochez et al. [4] uses GloVe [13] instead of the Skip-gram or Continuous Bag-of-Words (CBOW) algorithms. The node features are also derived using graph walks, in this case generating an approximation of the personalized PageRank for each node. Furthermore, many different strategies based on network statistics such as predicate frequencies are applied in creating biases in these graph walks.

With regard to applications of embeddings to solve the problem of entity resolution, the works of Bordes et al. [2] and Nickel et al. [12] are particularly relevant. Bordes et al. [2] uses pre-existing *EqualTo* relations in the graph and uses this information to learn an embedding. The query of whether two entities x and y are the same then amounts to asking how likely it is that the triple x *EqualTo* y holds. Nickel et al. [12] uses only the information in the embedding to decide if the entities x and y are equal by leveraging the similarity between their vector representations, with the similarity between entities x and y defined as the heat kernel $k(x, y) = e^{-||x-y||_2^2/\delta}$, where δ is a user-given constant and $||x - y||_2^2$ is the squared euclidean distance between x and y.

3 Data Sets

Historical data have their own peculiarities and difficulties, such as the fact that often information is approximate: we do not know when someone was born exactly and there exist many variations of how to write down a certain name. Over time errors accumulate while historians process the data, such as duplicate records for convenient searching in a file system being faithfully copied and digitised. This can make it much harder to extract information. Often some process of entity resolution is necessary first. In this work we use two historical data sets. Each describes a different aspect of the Dutch Golden Age:

– *The City Archives of Amsterdam* is a massive collection of registers, acts and books from the 16^{th} century up to modern times[1]. The original data are in the form of handwritten books that have been scanned and digitised by hand. Often more information is stored in the original form than was transcribed. Fully digitising all information is an ongoing process, often done by volunteers. For this project we made use of a subset collected from three different registers: Burial, Prenuptial Marriage and Baptism. The burial register does not describe who was buried where, but simply records the act of

[1] https://archief.amsterdam/indexen

someone declaring a deceased person. To this end, it mentions the date and place of declaration and references two persons, one of whom is dead. Sadly, it does not tell us which one of the two has died. The prenuptial marriage records tell us the church, religion and names of those who are planning to get married. It also mentions names of previous marriages if applicable. The baptism register mentions the place and date of where a child was baptised. It does not tell us the name of the child, only the names of the father and mother. Lastly the above records were combined with a subset from the Ecartico [2] data set, which is a comprehensive collection of biographical data from more well known people from the Dutch golden age. Figure 1 shows the graph representations of a record in each register. Note that these records can be linked to each other by sharing a literal node, in this case a name or date field, as shown in figure 2.

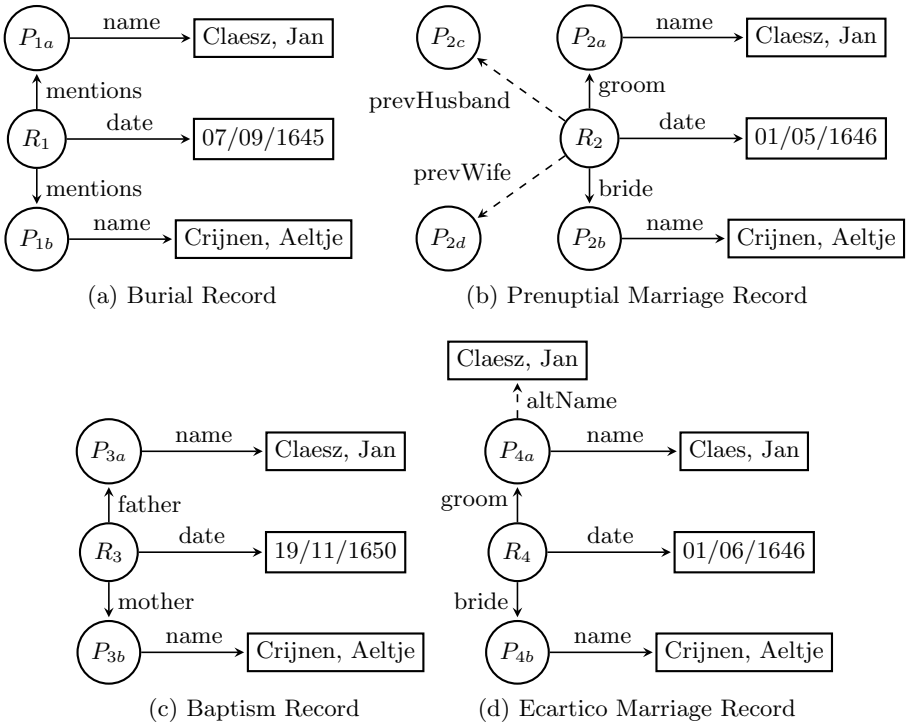


Fig. 1. Each registry in the Amsterdam City Archives has a number of records $(R_1, \ldots, R_n)$, all of which are associated with two or more persons $(P_{1a}, \ldots, P_{1z})$. These persons can have attributes for themselves, in this example we only use their full name. Dotted lines are optional relationships.

[2] http://www.vondel.humanities.uva.nl/ecartico

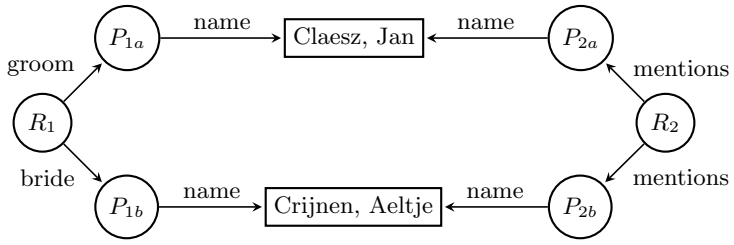


Fig. 2. Records can be linked together through literal nodes. In the above example a record in the prenuptial marriage registry is linked to a record in the burial registry.

– *Onstage*[3] is a data set about Amsterdams most prominent public theatre venue: the *Schouwburg*. It contains information from the year 1638 until today about plays, their creators and when these plays were shown. Additional information such as daily revenues is available but not used. Figure 3 shows a slightly simplified graph representation of the Onstage data set. Note that using different traversing strategies the neighborhood of, say, an author can in- or exclude the language of plays he has written. This has an effect on the resulting embedding, for instance when a user wants to make sure the language of an author is taken into consideration. In the case of Onstage we are not as interested in entity resolution as there are no duplicates present. Instead, we are interested in exploring the effects of different weights in the node neighborhood creation process.

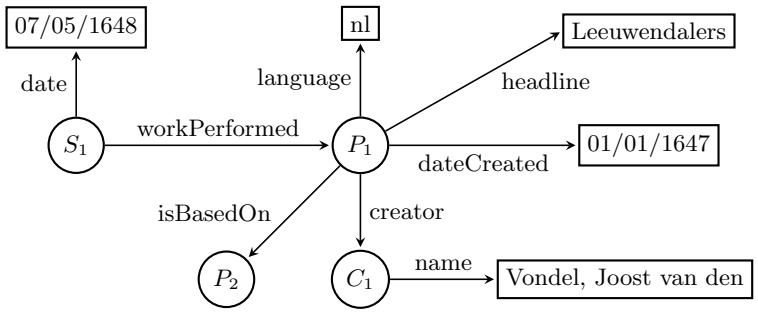


Fig. 3. Onstage general structure. Nodes labeled as S_i represent days on which certain works were performed. Nodes labeled as P_i are plays while those labeled as C_i represent authors.

In the official RDF specification, multiple literal nodes with the same value can coexist. This can be detrimental to the creation of network neighborhoods

[3] http://www.vondel.humanities.uva.nl/onstage

when literal nodes are the sole way of interconnecting the entities we are interested in. For this reason we slightly deviate from the standard and only allow one literal node to exist for each unique value. This also has the side effect of greatly reducing the size of the graph in many cases.

4 Approach

Inspired by the positive results reported by Pennington et al. [13] of GloVe over Word2Vec, our approach is to leverage the knowledge of expert historians to tailor an embedding of an RDF graph for application specific purposes such as entity resolution. An RDF graph can be viewed as a set of triples of the form $\langle subject, predicate, object \rangle$, where subject and object are represented by nodes in the graph and the predicate is represented by the edge between these nodes. The interpretation of such a triple is that the elements represented by the object and subject are related to each other by the relation represented by the predicate. Therefore we allow a user to provide the system with a list of predicates and associated weights. These weights represent the experts' opinion on the importance of predicates for some specific task. For example, in the case of entity disambiguation, the expert could determine that the name of a person is more important in relating him/her to another person than, say, a birth date, which can be inaccurate in historical sources. Like Cochez et al. we use an adaptation of the Bookmark Coloring Algorithm to generate derived features for each node in an RDF graph. However, due to the application specific nature and unique properties of each RDF graph, we experiment with different graph walk strategies in combination with the supplied weights given by an expert instead based on graph statistics. Furthermore we have developed our own code for calculating the embedding vectors, which allowed us to experiment with different types of stochastic gradient descent in order to determine if better minimum can be found and/or in fewer iterations.

5 Generating Network Neighborhoods

Let $G = (V, E)$ be a given RDF graph. As in Node2Vec, we define $N_S(i) \subset V$ as the neighborhood of node $i \in V$ generated through a neighborhood sampling strategy S. The objective is to create an embedding of size $|V| \times d$ where d is the (user defined) number of dimensions. In order to generate derived features for the nodes in an RDF graph we create a neighborhood for each node in the graph. This neighborhood consists of a set of nodes that can be reached from a certain starting node. The value given to each node in the neighborhood depends on the distance from the starting node and the network structure. Having multiple paths to the same node will increase this value. The output of the neighborhood generation algorithm is an (ideally sparse) co-occurrence matrix X of size $|V| \times |V|$ where X_{ij} is a non-zero positive real number iff node j occurs in the neighborhood of node i. The co-occurrence matrix X is then used as input for the GloVe algorithm.

The node neighborhoods themselves are generated using the *bookmark coloring algorithm* (BCA) [1]. A useful feature of BCA which distinguishes it from random walks is that the former is not random, which helps to reduce the number of random components in the system. The general idea of BCA is to consider the neighborhood of a node as an approximation of the personalized PageRank for that node. A useful analogy is imagining dropping a unit amount of paint on a given starting node. A fraction α remains on the node and the fraction $1 - \alpha$ is distributed among neighboring nodes using one of the strategies mentioned below. When after a certain point the amount of paint falls under ϵ the paint is no longer distributed. This means that even in the case of loops, the algorithm will eventually terminate after running out of paint. The user can tweak ϵ in order to get a smaller or larger neighborhood. The distance weight can be adjusted with α. Values in the range $[10^{-1}, 10^{-3}]$ for α and $[10^{-2}, 10^{-4}]$ for ϵ seem to work best for various applications. Having a very small ϵ can cause X to be too dense, which causes memory issues.

The RDF graph can be traversed in multiple ways to generate different neighborhoods depending on the wishes of the user and underlying structure of the graph. For some graphs, such as the Amsterdam City Archives graph, literal nodes, such as those representing names, act as hubs linking together concepts from multiple sub-graphs that represent for example different data sets. In other cases the edges are distributed much more equally among nodes. We have experimented with three different strategies of neighborhood sampling:

- *Undirected Weighted:* Ignore the edge directions and distribute paint based only on the weights given by the domain expert. This means that all incoming and outgoing edges are considered with the exception of the edge that leads to the previous node. This can be useful if the direction of edges in the network prevents certain important nodes from being visited from some starting nodes.
- *Directed Weighted:* Distribute paint among all outgoing edges according to weights given by a domain expert. Sometimes this process is reversed by, for instance, reversing all edges in the graph and distributing paint again. This creates a node neighborhood that contains all nodes leading up to and out of a starting node up to a certain distance. The reasoning behind this is that it is analogous to having a symmetric window when creating word embeddings [4].
- *Directed Weighted Literal:* This strategy behaves the same as directed weighted, however on literal nodes it follows incoming edges of the same type used for arriving on the literal node. This can be useful when multiple edges of different types are pointing to the same literal node. The intuition is that when arriving on a node representing a date via a *dateOfBirth* relationship we are only interested in other nodes that also use that same relationship.

6 Embedding Optimization

For generating the final embedding we make use of the GloVe [13] algorithm. Originally designed for natural language processing, it has also been used by Michael Cochez et al. [4] for embedding nodes of an RDF graph.

GloVe uses equation 1 as its loss function. Each node is associated with two vectors w and $\bar{w}$ representing coordinates in the embedding, and two bias terms b and $\bar{b}$. The vector and bias term w and b are used when the node is evaluated in its context, while $\bar{w}$ and $\bar{b}$ are used when a node is evaluated as the context of another node. Then, for some combination of nodes i and j, the similarity between vectors w_i and $\bar{w_j}$ is measured by their dot product and compared to the observed co-occurrence value X_{ij}. The function $f(.)$ acts to curb the influence of high values for X_{ij}, diminishes the effect of making mistakes for low values of X_{ij} and as a safeguard for when $X_{ij} = 0$. Finally, the output vector for a given node is $\frac{w+\bar{w}}{2}$. The bias terms $b, \bar{b}$ are dropped and not used.

$$J = \sum_{i}^{|V|} \sum_{j}^{N_S(i)} f(X_{ij})(w_i^T \bar{w}_j + b_i + \bar{b}_j - \log X_{ij})^2 \tag{1}$$

For minimizing J, we have experimented with several gradient descent algorithms in order to determine which one works best in the setting where we use GloVe in conjunction with a co-occurrence matrix created from our historical RDF data:

- *Adagrad:* [5] This algorithm is used by Pennington et al. [13] to train GloVe for word embeddings. Adagrad adapts the learning rate for each parameter by dividing the learning rate by the sum of squares of the past gradients. It does, however, suffer from the fact that the accumulated gradients cause the learning rate to shrink each step which can inhibit further learning.
- *Adadelta:* [17] An extension of Adagrad aiming to reduce the problem of the monotonically decreasing learning rate. Instead (among other things) it stores a decaying average of previous squared gradients.
- *Adam:* [9] Like Adadelta, Adam stores a decaying average of the previous squared gradients. However, it also keeps a decaying average of the past gradients. These act as approximations of the second and first moments of the gradients respectively. In our experiments Adam yielded unstable behavior. This could somewhat be corrected by tweaking the ϵ parameter. The comparison below uses $\epsilon = 10^{-8}$ as mentioned in the original paper.
- *AMSgrad:* [15] The last method we used, AMSgrad adapts the first moment to use the maximum of all past gradients instead of a decaying average. The intuition behind this is that large and informative gradients occur only rarely, and are otherwise 'forgotten' by the decaying average.

Each algorithm is stopped as soon as the difference between iterations falls below 10^{-5}. As shown in figure 4, our experiments show that AMSgrad performs best on both the Amsterdam City Archives and Onstage, converging in the least

time and to the best or equally good minimum. A possible explanation for this is that only certain combinations (i, j) of nodes are informative and have a large gradient. AMSgrad is best able to exploit this situation by storing these large gradients without decaying them over time. Adam has issues converging in our case. Preliminary tests have shown that setting Adams ϵ parameter to 0.1 or 1 can have beneficial effects. In the comparison, all algorithms use their default parameters as defined by their respective papers.

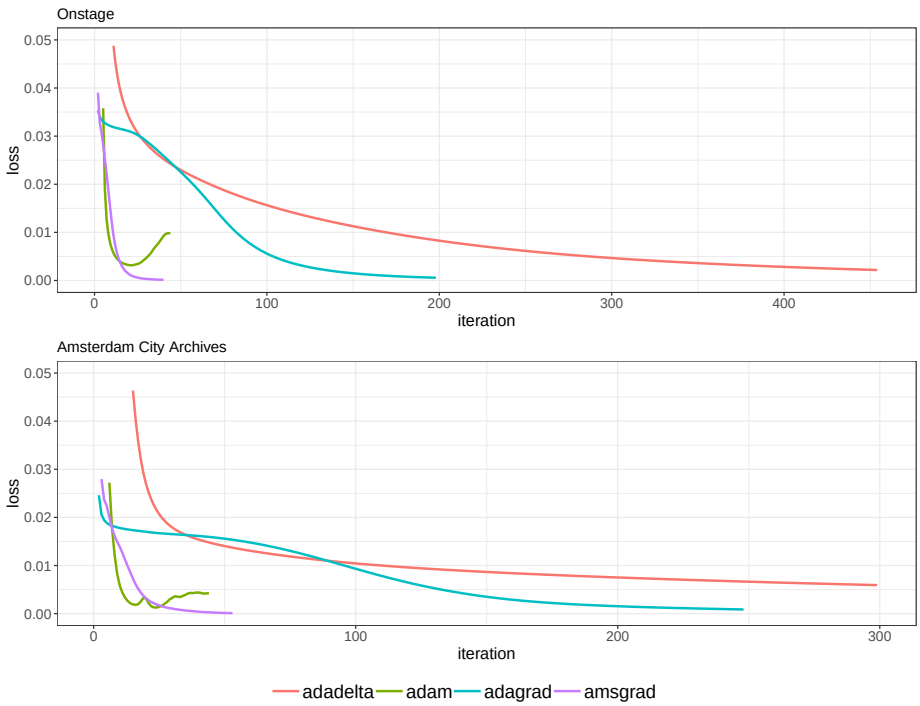


Fig. 4. A comparison of stochastic gradient descent methods. The Y axis is the normalized loss function J (divided by the number of co-occurrences) and the X axis is the number of iterations. Note that, in this case, the best performing in both cases is AMSgrad. Which converges in fewer iterations with equal or minimum cost.

These results are encouraging enough that it could be worth the effort to use AMSgrad instead of Adagrad in other applications when creating embeddings from graphs.

7 Implementation Details

The entire system has been written in Java as this makes it much easier to integrate into the Java based multi-agent system of the Golden Agents project.

Several libraries are used for specific tasks: we use Jena[4] for loading in the RDF data, and we use Grph[5] (high performance graph library) for representing the graph in memory as sequential integers which can be stored densely in memory. Grph also allows for pre-processing of the graph for generating a node-neighbor data structure beforehand which can be reused while iterating over each node in the graph. The independent nature of doing the graph walks without changing the underlying graph also allows us to run many walks in parallel.

The stochastic gradient descent algorithms are all written in pure Java and is based on the C code written at Stanford[6]. The batch size is 1, so at each co-occurrence we encounter the parameters are updated. Since each co-occurrence only updates a tiny fraction of all available parameters the issue of collisions is minimized and grows less likely as the number of nodes increases. This step can also be done in parallel, with each thread handling a random sample of the nodes each iteration. This random sampling stabilizes the output of the code as it counteracts the tendency to end up in local minima due to the randomized initial starting conditions.

Finally, the last optional step of performing PCA to reduce the number of dimensions in the embedding is done using netlib[7], a Java wrapper for low-level vector and matrix operations. We perform PCA in the end to reduce the size of the resulting embedding without losing too much information. We default to the minimum number of principal components that are necessary to explain 95% of the variance. At times this can lead to a reduction of 200 to just 4 dimensions, which somewhat offsets the need to choose the right number of dimensions d beforehand.

All our code is open source and available at our github repository[8].

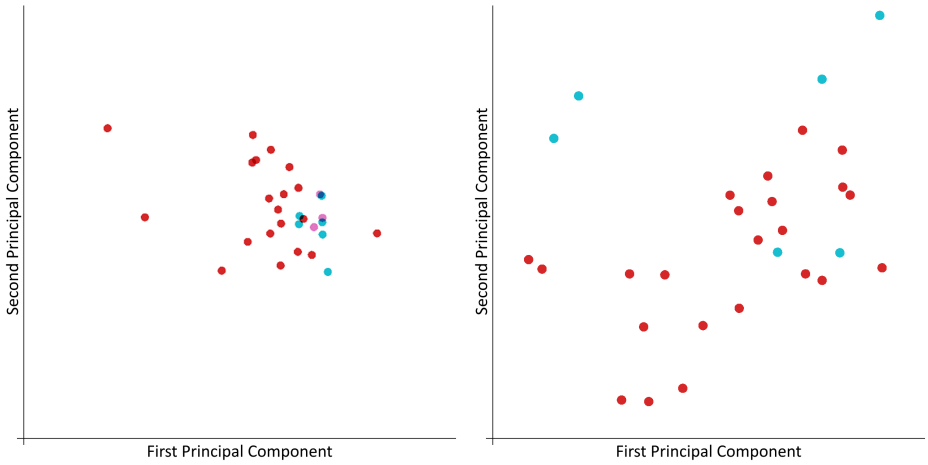
8 Preliminary Results

With the help of an embedding visualization tool[9] we can show some of our preliminary results. Figure 5 shows how we can improve the clustering of persons in the Amsterdam City Archives by using the right graph walk strategy for the problem. Since the Amsterdam City Archives is an agglomeration of multiple data sets which are interconnected via literal nodes, the *Undirected* strategy generates more representative neighborhoods by traversing more of the graph

[4] https://jena.apache.org
[5] http://www.i3s.unice.fr/ hogie/software/index.php
[6] https://github.com/stanfordnlp/GloVe
[7] https://github.com/fommil/netlib-java
[8] https://github.com/Jurian/graph-embeddings
[9] https://projector.tensorflow.org

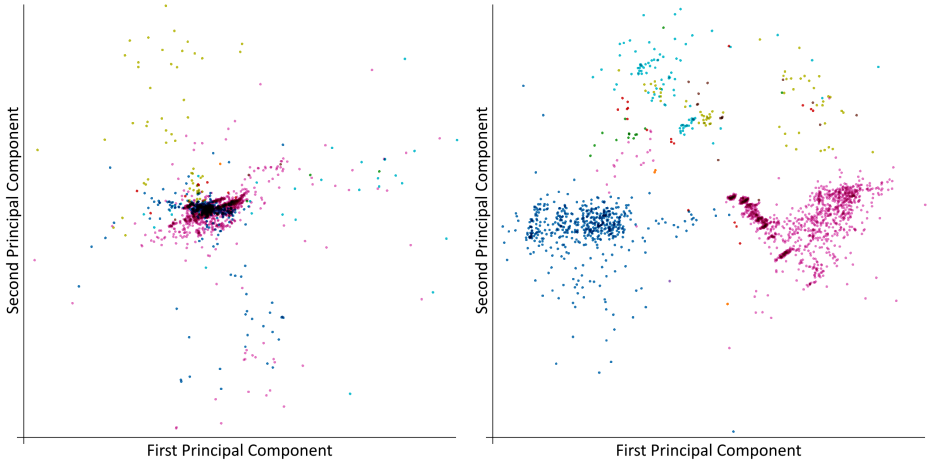
while at the same time not stopping when it encounters a node with solely outgoing or incoming edges at one of the record-nodes.

Figure 6 shows how with the choice of weights we are able to influence an embedding of the Onstage data set. For example, say a domain expert is interested in an embedding which emphasises the language of a play, in this case they can easily increase the weight associated with that relationship as shown in table 1. With the increased weight on language it becomes much easier to cluster plays based on this information. In order to check whether plays have indeed clustered together based on language we apply a model-based clustering algorithm [6] on the minimum number of principal components necessary to explain 95% of the variance. This results in 9 and 4 components for identical weights and a high language weight respectively. Tables 2 and 3 show the results of a discriminant analysis [7] of plays with the class label being the language of a play. This shows that we can achieve predictable results from a certain choice of weights.



(a) Directed (b) Undirected

Fig. 5. PCA analysis of a subset of a particularly frequent name in the Amsterdam City Archives: Jan Jansen. Each data point represents the mention of a person named Jan Jansen in some record of the archives. Pink points are from the baptism archive, cyan points are from the burial archive, while red points are from the prenuptial marriage archive. The three pairs of cyan points on the right are known duplicates in the burial archives. Note how they are much better distinguished using the undirected graph walk strategy. Both directed strategies proved insufficient for generating overlapping contexts. In this example all weights for relations are set to 1.



(a) No preference for any relation type (b) Preference for language of a play

Fig. 6. PCA analysis of plays in the Onstage data set. The panel on the right shows how plays are better clustered by language (noted by their color) after we give a higher weight to the language relationship during the neighborhood creation process, with all other settings being equal. See table 1 for exact weights used.

Table 1. The weights used to generate the panels in figure 6.

Relationship	Weight figure 6a	Weight figure 6b
http://schema.org/creator	1	1
http://schema.org/isBasedOn	1	1
http://schema.org/inLanguage	1	100

Table 2. Discriminant analysis classification result for clustering plays in the Onstage data set based on their language. This result was obtained using the left-hand weights in table 1. The row represents the class, the column represents the cluster. Classification error is 0.33. These results were obtained using the `MclustDA` method in R [7].

Class	da	de	en	es	fr	it	la	nl	sv
da	**1**	0	0	0	0	0	0	2	0
de	1	**12**	1	0	0	0	1	78	0
en	0	0	**10**	0	0	0	0	8	0
es	0	0	0	**29**	0	0	0	33	0
fr	0	1	3	9	**54**	1	2	448	0
it	0	0	0	0	1	**8**	0	6	0
la	0	0	0	0	2	0	**4**	8	0
nl	1	12	9	9	16	9	3	**1226**	0
sv	0	0	0	0	0	0	0	0	**1**

Table 3. Discriminant analysis classification result for clustering plays in the Onstage data set based on their language. This result was obtained using the right-hand weights in table 1. The row represents the class, the column represents the cluster. Classification error is 0.025. These results were obtained using the `MclustDA` method in R [7].

Class	da	de	en	es	fr	it	la	nl	sv
da	**1**	0	0	0	2	0	0	0	0
de	0	**90**	1	0	0	0	2	0	0
en	0	0	**10**	0	0	4	4	0	0
es	0	0	0	**62**	0	0	0	0	0
fr	1	0	0	0	**502**	1	3	11	0
it	0	0	1	0	0	**13**	0	1	0
la	0	1	0	0	0	0	**13**	0	0
nl	0	2	0	0	4	0	13	**1266**	0
sv	0	0	0	0	0	0	0	0	**1**

9 Conclusions and Further Research

In this paper we described our method of embedding nodes from an RDF graph, with some specific purpose in mind, aided by expert knowledge. We have shown that choosing the correct graph walking strategy matters for specific data sets, i.e. specific graph structures. We have also shown that using AMSgrad instead of Adagrad can give a performance boost when optimizing the loss function of GloVe, at least when using it to embed nodes in an RDF graph. Finally we show that we can tailor the weights to obtain task specific embeddings. Of course, many improvements and additions can yet be made and we will outline some of them here.

Many historical sources have a degree of uncertainty when it comes to information such as birth- and death days. Currently, when two sources disagree on some given date the graph walker will miss that particular bit of information. The same goes for names, these are often spelled in many different varieties and/or languages. Giving the graph walker the ability to traverse the graph while taking these 'weaker' relationships into account will likely help to connect entities that were previously unconnected.

Furthermore we plan to integrate these tailored embeddings into the Golden Agents multi-agent system. This will take the form of an agent which encapsulates some embedding. This agent can then answer queries from other agents such as 1) nearest neighbors for a certain node to give extra context and 2) split a given set of nodes into several clusters based on their positions in the embedding.

References

1. Berkhin, P.: Bookmark-coloring algorithm for personalized pagerank computing. Internet Mathematics **3**(1), 41–62 (2006)

2. Bordes, A., Glorot, X., Weston, J., Bengio, Y.: A semantic matching energy function for learning with multi-relational data. Machine Learning **94**(2), 233–259 (2014)
3. Brouwer, J., Nijboer, H.: Golden agents. a web of linked biographical data for the dutch golden age. In: CEUR Workshop Proceedings. vol. 2119, pp. 33–38 (2018)
4. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Global rdf vector space embeddings. In: International Semantic Web Conference. pp. 190–207. Springer (2017)
5. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research **12**(Jul), 2121–2159 (2011)
6. Fraley, C., Raftery, A.E.: Mclust: Software for model-based cluster analysis. Journal of classification **16**(2), 297–306 (1999)
7. Fraley, C., Raftery, A.E.: Model-based clustering, discriminant analysis, and density estimation. Journal of the American statistical Association **97**(458), 611–631 (2002)
8. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864. ACM (2016)
9. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
10. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
11. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013)
12. Nickel, M., Tresp, V., Kriegel, H.P.: A three-way model for collective learning on multi-relational data. In: ICML. vol. 11, pp. 809–816 (2011)
13. Pennington, J., Socher, R., Manning, C.: Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). pp. 1532–1543 (2014)
14. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710. ACM (2014)
15. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond. arXiv preprint arXiv:1904.09237 (2019)
16. Ristoski, P., Paulheim, H.: Rdf2vec: Rdf graph embeddings for data mining. In: International Semantic Web Conference. pp. 498–514. Springer (2016)
17. Zeiler, M.D.: Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)