Consensus Embeddings for Networks with Multiple Versions

Mengzhen Li and Mehmet Koyutürk

Department of Computer and Data Sciences
Case Western Reserve University, Cleveland, OH, USA

Abstract. Machine learning applications on large-scale network-structured data commonly encode network information in the form of node embeddings. Network embedding algorithms map the nodes into a low-dimensional space such that the nodes that are "similar" with respect to network topology are also close to each other in the embedding space. Many real-world networks that are used in machine learning have multiple versions that come from different sources, are stored in different databases, or belong to different parties. Due to efficiency or privacy concerns, it may be desirable to compute *consensus embeddings* for the integrated network directly from the node embeddings of individual versions, without explicitly constructing the integrated network. Here, we systematically assess the potential of consensus embeddings in the context of processing link prediction queries on user-chosen combinations of different versions of a network. For the computation of consensus embeddings, we use linear (singular value decomposition) and non-linear (variational autoencoder) dimensionality reduction methods. Our results on a large selection of protein-protein interaction (PPI) networks (eight versions with 255 potential combinations) show that consensus embeddings enable real-time processing of link prediction queries on user-defined combinations of networks, without requiring explicit construction of the integrated network. We observe that linear dimensionality reduction delivers better accuracy and higher efficiency than non-linear dimensionality reduction. We also observe that the performance of consensus embeddings is amplified with increasing number of networks in the database, demonstrating the scalability of consensus embeddings to growing numbers of network versions.

Keywords: node embedding, dimensionality reduction, link prediction

1 Introduction

Large-scale information networks are becoming ubiquitous. Mining knowledge from these information networks has become very popular in a broad range of applications. Learning a representation of networks is useful for many network analysis applications[16], including social network analysis[10, 24] and bioinformatics [2, 21].

With the increase in the quantity and variety of network datasets, effective integration of different data sources is becoming a popular and challenging task for researchers [19]. Many networks have multiple versions, as different data providers may gather their data from different sources, some of the data may not be shared due to privacy concerns [11], or the data that is available may evolve over time.

An important task in analyzing integrated networks is the computation of node embeddings, i.e. learning low-dimensional representation of integrated networks[24, 18]. Node embeddings aim to map each node in the network to a low dimensional vector representation to extract features that represent the topological characteristics of the network. Many techniques are developed for this purpose [8, 1, 22, 15], and these techniques are shown to be effective in addressing problems such as link prediction [23, 3], node classification [4], and clustering [17].

Different versions of a network have the same set of nodes and different sets of edges. These different sets of edges may represent identical semantics but different sources (e.g., protein-protein interaction (PPI) networks obtained from different databases) or different semantics (e.g., physical PPIs vs. genetic interactions). In many settings, it may not be possible or desirable to superpose multiple versions of a network. For example, in integrated querying of networks from multiple databases, computation of embeddings for all possible combinations may not be feasible or efficient [6]. Cho et al.[5] develop a method that uses random walk diffusion states of individual networks to compute the node embedding of the integrated network. In their method, node embeddings are learned from multiple $n \times n$ matrices of probabilities. However, the learning has to be performed at query time. From the perspective of efficiency and real-time query processing, computation of node embeddings at query time is not desirable. A potentially more efficient approach is to use the node embeddings of different versions to compute the embeddings of the integrated network (as opposed to computing embeddings using the integrated network).

Motivated by this observation, we introduce the notion of "consensus embeddings" as node embeddings for the integrated network that are computed from the embeddings of separate versions. To compute consensus embeddings, we use linear (singular value decomposition) and non-linear (variational autoencoder) dimensionality reduction. Using multiple versions of protein interaction networks, we systematically assess the accuracy and efficiency of concensus embeddings in the context of combinatorial link prediction queries. Our results show that use of consensus embeddings in processing link prediction queries significantly improves computational efficiency, without significantly compromising the accuracy of link prediction.

2 Methods

2.1 Node Embedding

Node embedding aims to learn a low-dimensional representation of nodes in networks[16]. Given a graph $G = (V, E)$, a node embedding is a function $f : V \rightarrow R^d$ that maps each node $v \in V$ to a vector in R^d where $d \ll |V|$. A node embedding method computes a vector for each node in the network such that the proximity in the embedding space reflects the proximity/similarity in the network.

In the last few years, many methods [8, 1, 22, 15] have been developed to compute node embeddings in a given network. Methods usually differ in terms of how they formulate the similarity between nodes (or the objective function that specifies the correspondence between the embedding and network topology). Node embedding methods can also be roughly divided into community-based approaches and role-based approaches [16]. Community based approaches aim to preserve the similarity

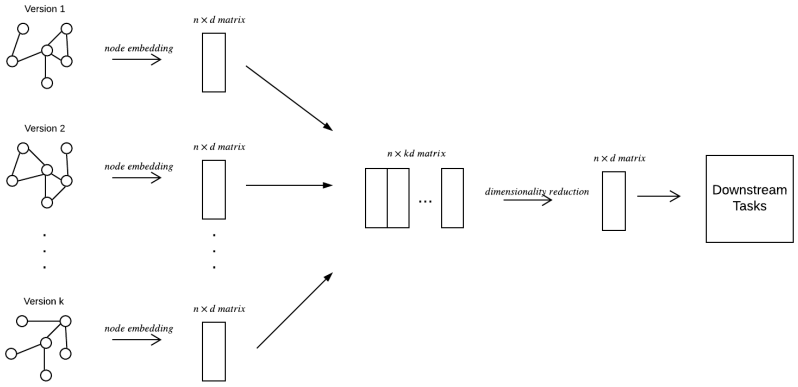


Fig. 1: **The proposed framework for the computation of consensus embeddings using dimensionality reduction methods.** The graphs labeled Version i represent multiple versions of a network with a fixed node set and different (possibly overlapping) edge sets. The objective is to compute node embeddings for the network obtained by superposing these versions. At the absence of the integrated network, we compute the consensus embedding by computing separate embeddings for each version and then using dimensionality reduction to compute a common reduced-dimensional space for these embedding spaces. Finally, we use the resulting consensus embeddings to perform downstream machine learning tasks on the integrated network.

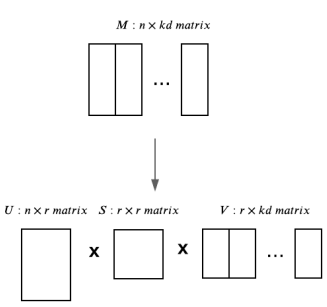
of the nodes in terms of the communities they induce in the network. In contrast, role based approaches aim to capture the topological roles of the nodes and map nodes with similar topological roles close to each other in the embedding space. As representatives of these different approaches, we here consider node2vec [8] as a community-based approach and role2vec [1] as a role-based approach.

2.2 Consensus Embeddings

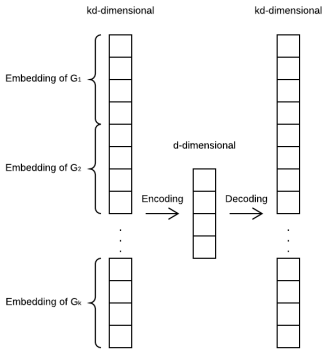
In this section, we formalize the problem of integrating multiple networks and computing node embeddings. Let $G_1 = (V, E_1)$, $G_2 = (V, E_2), ..., G_k = (V, E_k)$ be k versions of a network with the same set of nodes and different sets of edges, i.e., all the versions have the same set of n nodes. We consider the integration of these k networks through superposition of their edges, i.e., we define the integrated graph as $G = (V, E)$, where $E = \bigcup_{i=1}^{k} E_i$).

Assume that d-dimensional node embeddings X_i for G_i are given for $1 \leq i \leq k$, where X_i are $n \times d$ matrices and $X_i(j)$ is a d-dimensional vector representing the embedding of node $v_j \in V$ according to graph G_i. Our objective is to use X_i to compute d-dimensional node embeddings X_c for G, without using any other information on each of the G_is or G. We call X_c a *consensus embedding*.

This framework is illustrated in Figure 1, in which $node\ embedding$ can be any method for the computation of node embeddings (we here use node2vec and



(a) SVD(Singular Value Decomposition)



(b) Autoencoder

Fig. 2: Illustration of dimensionality reduction methods used to compute consensus embeddings.

role2vec), and $dimensionality\ reduction$ can be any dimensionality reduction method (we here use SVD or variational autoencoder).

2.3 Computing Consensus Embeddings

The input to the computation of consensus embeddings is k $n \times d$ matrices $X_1, X_2, ..., X_k$. To integrate these embeddings, we first create an $n \times kd$ matrix X by concatenating these k matrices. We then use dimensionality reduction on this matrix to compute an $n \times d$ matrix X_c, which represents the consensus embedding for G.

Singular Value Decomposition(SVD): Singular Value Decomposition(SVD) is a matrix decomposition method for reducing a matrix to its constituent parts. The singular value decomposition of an $m \times p$ matrix M, whose rank is r, is a factorization of the form USV^T, where U is an $m \times r$ unitary matrix, S is an $r \times r$ diagonal matrix, and V is an $p \times r$ unitary matrix. S is a diagonal matrix and the diagonal values of S are called the singular values of M.

Letting $M = X$ in this formulation, we obtain $n \times r$ dimensional matrix U, $r \times r$ dimensional matrix S, and $nd \times r$ dimensional matrix V, where r denotes the rank of X and $X = USV^T$. Our objective is to compute a d-dimensional matrix X_c such that $X_c X_c^T$ approximates XX^T well. If we set our objective as one of choosing $n \times kd$ dimensional matrix Y with rank d to minimize the Frobenius or 2-norm of the difference $||X - Y||$, then the optimal solution is given by the truncation of SVD to the largest d singular values (and corresponding singular vectors) of X. Namely, let U', S', and V' denote the $n \times d$, $d \times d$, and $kd \times d$ matrices obtained by choosing the first d columns (also rows for S) of respectively U, S, and V. Then the matrix $Y = U'S'V'^T$ provides the best rank-d approximation to X. Consequently, V' provides an optimal mapping of the kd dimensions in X to d-dimensional space. Based on this observation, SVD-based dimensionality reduction sets

$$X_c^{(SVD)} = XV'^T, \tag{1}$$

i.e., it maps the kd-dimensional concatenated embedding of each node of the graph into the d-dimensional space defined by the SVD of X. Figure 2(a) shows the dimensions of matrices when computing the consensus embeddings via SVD.

Variational Autoencoder: An autoencoder is an unsupervised learning algorithm that applies backpropagation to obtain a lower-dimensional representation of data, setting the target values to be equal to the inputs. The use of a convolutional autoencoder for dimensionality reduction in the context of computing consensus embeddings is shown in Figure 2(b). As seen in the figure, the autoencoder is a neural network with kd inputs, each representing a column of the matrix X (i.e., a dimension in one of the k embeddings spaces). The layer(s) on the left (encoder) map these kd inputs to d latent features shown in the middle, which are subsequently transformed into the kd output by the layer(s) on the right (decoder). While training the network, each row of the matrix X (i.e., the embedding of each node) is used as an input and the respective output. The neural network is trained using this loss function:

$$L(X,Y) = \|X - Y\|_F^2, \tag{2}$$

where Y denotes the $n \times kd$ matrix whose rows represent the outputs of the network corresponding to the inputs that represent the rows of X. Thus the idea behind the variational autoenconder is to learn an encoding of the kd input dimensions into the d latent features (shown in the middle) such that the kd inputs can be reconstructed by the decoder with minimum loss. Observe that this loss function is identical to that of SVD; however, the use of neural networks provides the ability to perform nonlinear dimensionality reduction. Once the neural network is trained, we perform dimensionality reduction by retaining the d-dimensional output of the encoder that corresponds to each of the n training instances (rows of the matrix X or nodes in V). These n d-dimensional vectors comprise the matrix $X_C^{(VAE)}$, i.e, consensus embeddings of the nodes in V computed by variational autoencoder.

In our implementation, we use a convolutional autoencoder [14]. Same as a standard autoencoder, a convolutional autoencoder also aims to output the same vectors as the input. The convolutional autoencoder contains convolutional layers in the encoder part of the autoencoder. In every convolutional layter, there is a filter that slides around the input matrix to compute the next layer. Convolutional autoencoder also have pooling layers after each convolutional layer. In the decoder part, there are deconvolutional layers and unpooling layers that recovers the input matrix.

2.4 Link Prediction

Link prediction is an important task in network analysis [13]. Given a network $G = (V,E)$, link prediction aims to predict the potential edges that are likely to appear in the network based on the topological relationships between pairs of nodes. Link prediction can be supervised [7] or unsupervised [9]. For supervised link prediction, the known links serve as positive samples and disconnected pairs of nodes serve as negative samples. The embedding vectors of nodes are treated as feature vectors and used to train the classifiers [23]. For unsupervised link prediction, the distances between pairs of vectors can be used to predict the proximity between nodes in the network and thus predicts the potential edges by ranking the distances [3].

In our experiments, we use BioNEV [23] to test the performance of the link prediction accuracy of the consensus embeddings. It is a supervised method that aims to systematically evaluate embeddings. It outputs the AUC scores of the link predictions using the embeddings.

2.5 Processing Combinatorial Link Prediction Queries for Versioned Networks

Consider the following scenario: A graph database houses k versions of a network (as formulated at the beginning of this section). These k versions may either come from different resources (e.g., different protein-protein interaction databases) or represent semantically different types of edges between a common set of nodes (e.g., genetic interactions vs. physical interactions vs. functional association among human proteins). In this setting, a “combinatorial" link prediction query can be formulated as follows: The user chooses (i) a node $q \in V$, and (ii) a subset $S \subseteq \{G_1, G_2, ..., G_k\}$ of networks. The query seeks to identify the nodes that are most likely to be associated with the query node q based on the topology of the integrated network $G^{(S)} = (V, E^{(S)})$, where $E^{(S)} = \bigcup_{i \in S} E_i$. Such a flexible query framework is highly useful in the context of many applications, since the relevance and reliability of different network versions can be variable, and different users may have different needs and preferences.

The above framework defines a “combinatorial" query in the sense that a user can select any combination of networks to integrate. This poses a significant computational challenge as the number of possible combinations of networks is exponential in the number of networks in the database, i.e., the user can choose from $2^k - 1$ possible combinations of networks. As we discuss in Section 2.4, there are many different ways of processing the link prediction queries. Among existing approaches, embedding-based link prediction techniques method demonstrated success in the context of many applications [23, 12]. Furthermore, embedding based link-prediction can facilitate the development of effective solutions to the combinatorial challenge associated with combinatorial link prediction queries, because link prediction algorithms using node embeddings do not need to access to the network topology while performing link prediction. By computing and storing node embeddings in advance, it is possible to efficiently process link prediction queries while giving the user the flexibility to choose the combination of networks to integrate.

Two possible approaches to addressing the combinatorial challenge represent two ends of the pre-processing/storage vs. query runtime trade-off:

- **Exhaustive Pre-Computation:** Compute the embeddings for each possible combination, store those embeddings. When the user selects a combination, use the embeddings for that combination directly. This approach minimizes query processing time while maximizing storage and pre-processing cost.
- **Network Integration at Query Time:** Store the individual network versions in the database without computing any embeddings before query. When the user selects a combination, construct the integrated network, compute the embeddings, and then use the embeddings to process the query. This approach avoids computing and storing an exponential number of embeddings, but performs all computations during query processing.

Table 1: **The description and size of the human protein-protein interaction (PPI) networks used in our experiments.**

Version:	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8
Interaction Type:	Affinity Capture-MS	Affinity Capture-RNA	Affinity Capture-Western	Negative Genetic	Positive Genetic	Synthetic Growth Defect	Synthetic Lethality	Two-hybrid
# Edges:	13472	3160	6132	65369	13018	9295	6842	4202

Consensus embeddings provide an alternate solution that can render storage feasible while enabling real-time query processing for very large networks and large number of versions:

– **Consensus Embedding at Query Time:** Compute and store the embeddings for each network separately. When the user selects a combination, compute a consensus embedding for that combination and use it to process the query.

One important consideration in the application of this idea is the “inexact" nature of consensus embeddings, i.e., consensus embeddings may not adequately capture the information represented by the embeddings computed on the integrated network. In the following section, we perform computational experiments to characterize the inexact nature of consensus embeddings on the accuracy of link prediction. We also investigate the earnings provided by consensus embeddings in terms of the required computational resources in processing combinatorial link prediction queries.

3 Results and Discussion

In this section, we present comprehensive experimental results on versioned networks in link prediction and discuss the implications of the results.

3.1 Datasets:

In our computational experiments, we use protein-protein interaction (PPI) networks obtained from BioGRID [20]. PPI networks contain physical interactions and functional associations between pairs of proteins. The dataset we use contains multiple PPI networks separated based on experimental systems. Each network (version) contains a unique type of PPI (genetic or physical). The types of the interactions represented by each network version are shown in Table 1. In order to obtain multiple networks with the same set of nodes, we remove the nodes (proteins) that do not exist in all versions. After preprocessing, all versions have 1025 nodes and different numbers of edges ranging from 3160 to 65369. The type of PPI and the number of edges for each network are shown in Table 1.

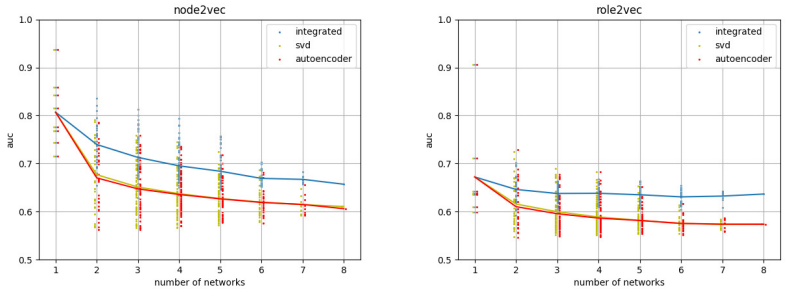
3.2 Accuracy of Link Prediction

We compare the link prediction performance of the node embeddings computed on integrated networks and consensus embeddings computed based on the embeddings of individual networks. We consider two embedding algorithms, Node2vec [8]

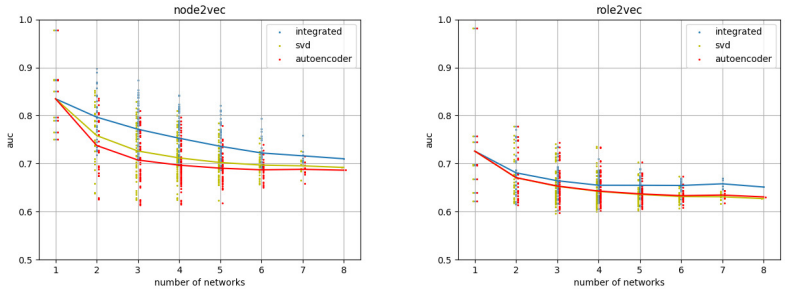
and Role2vec [1], and two methods for computing consensus embeddings, SVD and variational autoencoder. To assess link prediction performance, we use BioNEV[23], a Python package that is developed to assess the performance of various tasks that utilize network embeddings. Given a network and its node embeddings, BioNEV generates random training and testing sets to evaluate the link prediction performance of the embedding. BioNEV uses the known interactions as positive samples and randomly selects the negative samples. Both samples are split into a training set(80%) and a testing set(20%). For each node pair, BioNEV concatenates the embeddings of two nodes as the edge feature and then build a binary classifier. Using BioNEV, we obtain the area under ROC curve (AUC scores) for the link prediction performance of node embeddings generated using different methods.

Figure 3 shows the performance of consensus embeddings in link prediction compared with the performance of the integrated networks' embeddings. In each figure, the AUC of link prediction is shown as a function of the number of network versions. When there is a single version, the consensus embedding is identical to the embedding of the individual network. We observe that, on average, the accuracy of link prediction goes down with increasing number of versions that are integrated. However, the performance difference between embedding of integrated network and consensus embeddings becomes smaller with increasing number of versions. This observation suggests that the utility of concensus embeddings can be more pronounced for network databases with larger number of versions. We also observe that there is considerable variance of accuracy across different combinations with the same number of versions, indicating that some combinations of PPI types are more informative in predicting new PPIs as compared to other combinations.

As seen in Figure 3, accuracy of link prediction is improved with increasing number of dimensions in node embeddings. Importantly, with growing number of dimensions, the link prediction performance of consensus embeddings converge to that of the embeddings computed on the integrated network. Across the board, consensus embeddings computed using linear dimensionality reduction (SVD) deliver more accurate link prediction as compared to those computed using variational autoencoder. Since the edge set of the integrated network is a union of the edges of individual networks, the adjacency matrix of the integrated matrix can be approximated with a linear combination of the adjacency matrices of the individual networks. This might be the reason why linear dimensionality reduction performs better than neural networks. Finally, we observe that Node2vec delivers consistently more accurate link prediction as compared to Role2vec. The performance of consensus embeddings on larger numbers of networks is also better with Node2vec as compared to Role2vec. This is not surprising as Node2vec is based on communities in the network whereas Role2vec is based on roles. Integration of versions can change the topological features (e.g. degrees or paths) of the individual versions, and thus can have a stronger effect on the "roles" of nodes as compared to communities. Therefore, the embeddings computed via Role2vec are less robust to slight variations in network topology.



(a) number of dimensions = 16

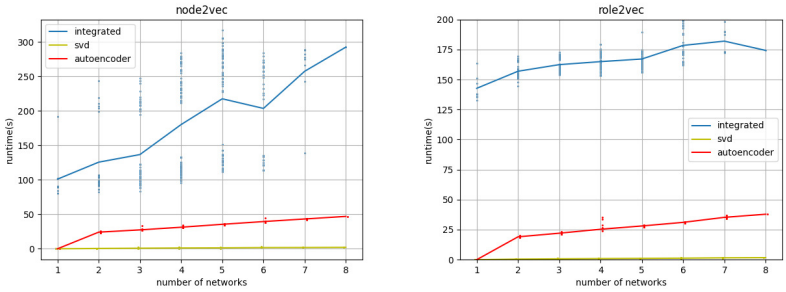


(b) number of dimensions = 64

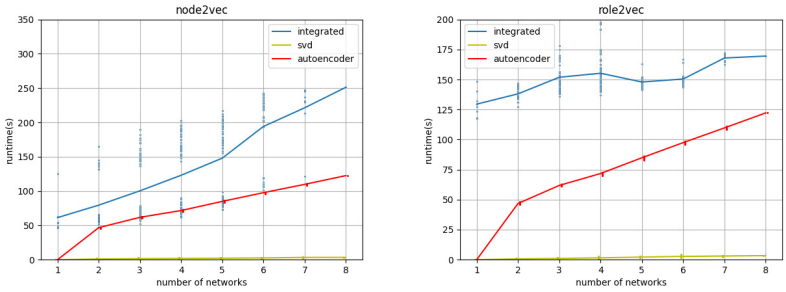
Fig. 3: **Accuracy of consensus embeddings in link prediction.** Results are shown for embedding methods Node2vec (left panels) and Role2vec (right panels). For each point k on the x axis, each point in the plot shows the area under ROC curve (AUC) of link prediction for a specific combination of k network versions. The lines show the average AUC across all combinations as a function of the network versions that are integrated. The blue, yellow, and red points/lines respectively show the accuracy provided by the embeddings computed directly on the integrated network, consensus embeddings computed using variational autoencoder, and consensus embeddings computed using SVD.

3.3 Computational Resource Requirements

In this section, we investigate whether consensus embeddings improve the efficiency of processing link prediction queries. For this purpose, we first compare query processing time for consensus embeddings computed using different methods (SVD and autoencoder) against embeddings computed at query time after integrating the combination of networks selected by the user. We use a high performance computing environment with a 2.2 GHz processor and 4GB memory. The results of this analysis are shown in Figure 4. As seen in the figure, processing queries using consensus embeddings drastically improves the efficiency of query processing. For both node2vec or role2vec, “Consensus Embedding at Query Time" using SVD enables processing of combinatorial link prediction queries in real time across the board, while integration of networks at query time requires orders of magnitude more time to process these queries. In most cases, "Consensus Embedding at Query Time" convolutional



(a) number of dimensions = 16



(b) number of dimensions = 64

Fig. 4: **Runtime of combinatorial link prediction queries using network embeddings.** The blue dots (for each combination) / curves (average of all combinations with the respective number of versions) show the query time corresponding to the “Network Integration at Query Time" approach described in Section 2.5, while the red and yellow dots/curves show the query time corresponding to the “Consensus Embedding at Query Time". Results are shown for two node embedding algorithms, Node2vec (left panel) and Role2vec (right panel), and two methods for computing consensus embeddings, SVD (yellow) and autoencoder (red). Each row shows a different number of dimensions for node embeddings.

autoencoder is also faster than "Network Integration at Query Time", but its performance degrades with increasing number of networks that are being integrated.

The runtime of computing an embedding increases as networks become denser, especially for node2vec. As seen in 4, the blue dots are separated into two groups for node2vec. This is because G_4 is extremely dense (see Table 1), making the integrated networks that contain G_4 also dense. Therefore, combinations that contain G_4 have a significantly higher query runtime as compared to those that do not contain G_4. Computation of consensus embeddings using SVD is more robust to this effect.

Next, we investigate the trade-off between the earnings in query runtime and pro-processing time/storage requirements. As discussed in Section 2.5, we consider three options for the processing of combinatorial link prediction queries. While the trade-off between storage/pre-processing vs. query runtime requirements for each of these approaches is intuitive, we also assess the performance of each approach in the context of this trade-off. The results of this analysis are shown on Table 2.

Table 2: **Assessment of the trade-off between pre-processing time, storage requirements, and query runtime for combinatorial link prediction on versioned networks.** Results are shown for a database of 8 network versions and 64-dimensional embeddings. SVD is used to compute consensus embeddings.

	Storage	Preprocess time	Query Runtime
Exhaustive Precomputation	180 MB	32676.624s	0s
Network Integration at Query Time	1.02 MB	0s	128.144 ± 60.627s
Consensus Embedding at Query Time Time	10.0 MB	490.354s	1.840 ± 0.745s

(a) Node2vec

	Storage	Preprocess time	Query Runtime
Exhaustive Precomputation	322 MB	38278.802s	0s
Network Integration at Query Time	1.02 MB	0s	150.113 ± 17.222s
Consensus Embedding at Query Time Time	5.53 MB	1036.038s	1.616 ± 0.822s

(b) Role2vec

As seen on Table 2, “Exhaustive Pre-Computation" makes query processing time extremely efficient since node embeddings are readily available during query processing with this approach. However, pre-processing time and storage requirements grow exponentially with the number of versions in the database, making this approach infeasible for practical applications. “Network Integration at Query Time" has zero pre-processing time, storage goes up linearly with the number of versions, but the query processing time is very slow because it needs to compute embedding at query time. “Consensus Embedding at Query Time" effectively balances this trade-off by as it requires pre-processing time and storage that grows linearly with the number of versions, but has always fast query processing time. In our experiments, there are 8 networks. However, in many practical settings, the number of networks can be large enough to render storage of all combinations infeasible.

4 Conclusion

In this work, we consider the problem of computing node embeddings for integrated networks derived from the multiple network versions. We define consensus embeddings as the node embeddings of the integrated network computing using the embeddings of individual versions. We test the performance of link prediction of the consensus embeddings and found that accuracy of consensus embeddings is close to the accuracy of embeddings computed directly from the integrated network. Our runtime analyses show that consensus embeddings are much more efficient than computing embeddings from the integrated network of multiple versions.

References

1. N. Ahmed, R. A. Rossi, J. B. Lee, X. Kong, T. L. Willke, R. Zhou, and H. Eldardiry. Learning role-based graph embeddings. *ArXiv*, abs/1802.02896, 2018.

2. S. K. Ata, Y. Fang, M. Wu, X.-L. Li, and X. Xiao. Disease gene classification with metagraph representations. *Methods*, 131:83 – 92, 2017.
3. A. Bojchevski and S. Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.
4. S. Cavallari, V. W. Zheng, H. Cai, K. C.-C. Chang, and E. Cambria. Learning community embedding with community detection and node embedding on graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 377–386, New York, NY, USA, 2017. Association for Computing Machinery.
5. H. Cho, B. Berger, and J. Peng. Compact integration of multi-network topology for functional analysis of genes. *Cell systems*, 3(6):540–548, 2016.
6. T. Cowman, M. Coşkun, A. Grama, and M. Koyutürk. Integrated querying and version control of context-specific biological networks. *Database*, 2020, 04 2020. baaa018.
7. H. R. de Sá and R. B. C. Prudêncio. Supervised link prediction in weighted networks. In *The 2011 International Joint Conference on Neural Networks*, pages 2281–2288, 2011.
8. A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.
9. T.-T. Kuo, R. Yan, Y.-Y. Huang, P.-H. Kung, and S.-D. Lin. Unsupervised link prediction using aggregative statistics on heterogeneous social networks. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 775–783, New York, NY, USA, 2013. Association for Computing Machinery.
10. J. Lin, L. Zhang, M. He, H. Zhang, G. Liu, X. Chen, and Z. Chen. Multi-path relationship preserved social network embedding. *IEEE Access*, 7:26507–26518, 2019.
11. Z. Ma, J. Ma, Y. Miao, and X. Liu. Privacy-preserving and high-accurate outsourced disease predictor on random forest. *Information Sciences*, 496:225 – 241, 2019.
12. K. Mallick, S. Bandyopadhyay, S. Chakraborty, R. Choudhuri, and S. Bose. Topo2vec: A novel node embedding generation based on network topology for link prediction. *IEEE Transactions on Computational Social Systems*, 6(6):1306–1317, 2019.
13. V. Martínez, F. Berzal, and J.-C. Cubero. A survey of link prediction in complex networks. *ACM Comput. Surv.*, 49(4), Dec. 2016.
14. J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In T. Honkela, W. Duch, M. Girolami, and S. Kaski, editors, *Artificial Neural Networks and Machine Learning – ICANN 2011*, pages 52–59, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
15. B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014.
16. R. A. Rossi, D. Jin, S. Kim, N. Ahmed, D. Koutra, and J. B. Lee. From community to role-based graph embeddings. *ACM Trans. Knowledge Discovery from Data (TKDD)*, 2019.
17. B. Rozemberczki, R. Davies, R. Sarkar, and C. Sutton. Gemsec: Graph embedding with self clustering. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '19, page 65–72, New York, NY, USA, 2019. Association for Computing Machinery.
18. X. Shen, Q. Dai, S. Mao, F. Chung, and K. Choi. Network together: Node classification via cross-network deep network embedding. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2020.
19. K. Shobha and S. Nickolas. Integration and rule-based pre-processing of scientific publication records from multiple data sources. In S. Satapathy, editor, *Smart Intelligent Computing and Applications*, pages 647–655, 2020.
20. C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. BioGRID: a general repository for interaction datasets. *NAR*, 34(suppl_1):D535–D539, 01 2006.
21. C. Su, J. Tong, Y. Zhu, P. Cui, and F. Wang. Network embedding in biomedical data science. *Briefings in Bioinformatics*, 21(1):182–197, 12 2018.

22. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077, 2015.
23. X. Yue, Z. Wang, J. Huang, S. Parthasarathy, S. Moosavinasab, Y. Huang, S. M. Lin, W. Zhang, P. Zhang, and H. Sun. Graph embedding on biomedical networks: methods, applications and evaluations. *Bioinformatics*, 36(4):1241—1251, February 2020.
24. J. Zhang, C. Xia, C. Zhang, L. Cui, Y. Fu, and P. S. Yu. Bl-mne: Emerging heterogeneous social network embedding through broad learning with aligned autoencoder. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 605–614, 2017.