Graphlets versus node2vec and struc2vec in the task of network alignment

Shawn Gu
Department of Computer Science and Engineering, Eck Institute for Global Health, and Interdisciplinary Center for Network Science and Applications (iCeNSA), University of Notre Dame
Notre Dame, Indiana

Tijana Milenković*
Department of Computer Science and Engineering, Eck Institute for Global Health, and Interdisciplinary Center for Network Science and Applications (iCeNSA), University of Notre Dame
Notre Dame, Indiana
tmilenko@nd.edu

ABSTRACT

Network embedding aims to represent each node in a network as a low-dimensional feature vector that summarizes the given node's (extended) network neighborhood. The nodes' feature vectors can then be used in various downstream machine learning tasks. Recently, many embedding methods that automatically learn the features of nodes have emerged, such as node2vec and struc2vec, which have been used in tasks such as node classification, link prediction, and node clustering, mainly in the social network domain. There are also other embedding methods that explicitly look at the connections between nodes, i.e., the nodes' network neighborhoods, such as graphlets. Graphlets have been used in many tasks such as network comparison, link prediction, and network clustering, mainly in the computational biology domain. Even though the two types of embedding methods (node2vec/struct2vec versus graphlets) have a similar goal – to represent nodes as features vectors, no comparisons have been made between them, possibly because they have originated in the different domains. Therefore, in this study, we compare graphlets to node2vec and struc2vec, and we do so in the task of network alignment. In evaluations on synthetic and real-world biological networks, we find that graphlets are both more accurate and faster than node2vec and struc2vec.

This work falls under the following submission types: "Novel research paper" and "Appraisal paper of existing methods and tools".

1 INTRODUCTION

Many complex systems can be modeled as networks [2, 23]. For example, social interactions between people can be modeled as social networks, and biochemical interactions between proteins inside the cell can be modeled as protein-protein interaction (PPI) networks. Modeling a system as a network allows us to consider the important interactions between entities (e.g., people, proteins, etc.), which can lead to deeper insights compared to analyzing each entity on its own.

An important and popular computational problem in the field of network science is network embedding [5]. The goal of network embedding is to represent each node in a network as a low-dimensional feature vector such that the network structure is preserved. The nodes' feature vectors can then be used in various downstream machine learning tasks. For example, in the task of node classification, given a network where labels are known only for some of the nodes, one can embed all nodes in a low-dimensional space and train a classifier to predict labels of the other nodes based on their feature vector similarities, i.e., closeness in the space, to the labeled nodes [6, 10, 26]. In the task of link prediction, after obtaining a feature vector for each node, one can calculate similarities between all node pairs, and then nodes with higher similarities will have higher probabilities of being linked [16, 37]. In the task of network clustering, the nodes' (or edges') feature vectors can be given as input to a clustering algorithm to group similar nodes (or edges) together [4, 13, 17, 20, 28].

Many network embedding methods *automatically learn* the features of nodes. That is, these methods formulate the problem of embedding as the optimization of some objective function. Two recent state-of-the-art methods, *node2vec* [10] and *struc2vec*[26], fall under this category. Intuitively, they use random walks to explore the extended neighborhood of a node and summarize it into the node's feature vector. To date, these approaches have been used in node classification, link prediction, and node clustering tasks, mainly in the social network domain [6, 10, 13, 17, 26, 37].

Other embedding methods *explicitly look* at the connections between nodes, i.e., the nodes' network neighborhoods, rather than trying to infer some features automatically through optimization. *Graphlets* fall under this category. Graphlets (Fig. 1) are Lego-like building blocks of complex networks, i.e., small subgraphs of a network (a path, triangle, square, etc.). Graphlets can be used to summarize the extended neighborhood of a node into a feature vector as follows. For each node, for each topological node symmetry group (formally, automorphism orbit), one can count how many times the given node touches each graphlet at each of its orbits. The resulting counts for all graphlets/orbits form the node's *graphlet degree vector (GDV)* [20]. Graphlets and nodes' (as well as edges') GDVs have been used extensively in many tasks, such as network comparison, network clustering, and link prediction, mainly in the computational biology (i.e., biological network) domain [8, 16, 20, 27–29, 36, 38].

Even though the methods that automatically learn node embeddings have a similar goal as graphlets – to obtain a feature vector of a node – to our knowledge no comparisons have been made between them, possibly because the two approach types have originated and have been used in the different domains (social versus biological networks). Even though recent studies on automatic learning-based embedding have recognized that graphlets can be seen as an alternative method for embedding, they have not compared against graphlets in their evaluations [6, 40]. To close this gap and merge the knowledge from the different domains, in this study

*To whom correspondence should be addressed

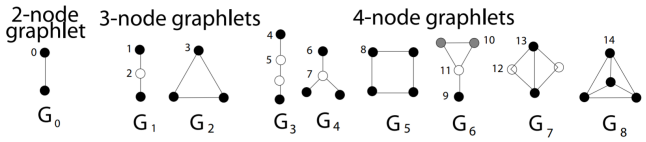

Figure 1: Illustration of all nine 2-4-node graphlets and their 15 node symmetry groups (i.e., automorphism orbits). In a given graphlet, nodes belonging to the same orbit are colored the same.

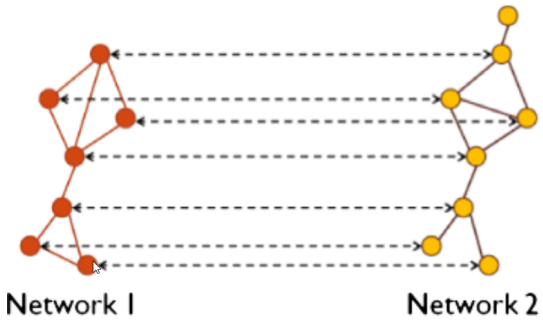

Figure 2: Illustration of network alignment (NA). NA aims to find a one-to-one (injective) mapping between nodes of the compared networks. Dotted lines represent nodes that are aligned to each other in this illustration. Note that NA as we define it in this study is called global NA, as it aims to map all nodes from the smaller of the two networks to nodes from the larger network. Another type of NA exists called local NA, which typically results in only small (local) network regions being mapped to each other [12, 19], but this is out of the scope of our study. Also, note that NA as we define in this study is called pairwise NA, as it aims to align two networks. Another type of NA exists called multiple NA, which can align more than two networks [32, 34], but this is out of the scope of our study.

we compare the two types of embedding methods to each other. Specifically, we compare graphlets to node2vec and struc2vec, and we do so in the task of network alignment (NA).

Intuitively, NA aims to find a node mapping between the compared networks that uncovers network regions of high similarity (Fig. 2) [7, 12, 19]. This allows for the transfer of functional knowledge between the similar (i.e., aligned) network regions. NA is also referred to as alignment-based network comparison, graph matching, graph deanonymization, and identity matching. This is because NA has been applied to many domains. For example, if we align the PPI network of baker's yeast, a well-studied species, to the PPI network of human, a poorly-studied species, we can infer the function of human proteins based on the function of their aligned partners in the yeast network. This was done in order to study human aging [9], which is otherwise difficult to do because of long life span and ethical constraints involving the human species. NA can also be used to deanonymize online social networks. For example, many Internet users are on multiple social media platforms, and thus, NA can be used to match identities (i.e., user accounts) across the different platforms [39]. On the other hand, NA can be used to study how to prevent such deanonymization attacks on potentially sensitive data, thus having privacy implications [22].

NA is computationally intractable, i.e., NP-hard, due to the NP-completeness of the underlying subgraph isomorphism problem [3]. So, heuristic algorithms need to be sought. These algorithms typically consist of two parts. First, they measure pairwise similarities between nodes from the networks being aligned. Then, they use an alignment strategy to quickly identify alignments that maximize some objective function, which often takes into account the total similarity over all aligned nodes (i.e., the goal is to align similar nodes to each other), plus potentially the amount of edges that are conserved under the given node mapping.

So, in our evaluation, we use each of graphlets (i.e., GDVs), node2vec, and struc2vec to quantify node similarities, plug those similarities into two established alignment strategies – WAVE [30] and SANA [18], and compare the results of the different node similarity measures under the same alignment strategy. We choose these two alignment strategies because they are recent and state-of-the-art methods, and also they are complementary to each other in the sense that they use different algorithmic paradigms (WAVE uses a seed-and-extend approach while SANA uses a search algorithm) [11]. We analyze synthetic networks originating from different random graph models as well as real-world biological networks. We align each of the networks to its randomly rewired (i.e., noisy) versions and test the robustness of each NA approach to noise in the data. Since the aligned networks have only a percentage of their edges different but they have the same nodes, we know which nodes should be mapped to which. So, we quantify NA quality by measuring node correctness – the percentage of nodes in the given alignment that are correctly mapped. Note that while aligning a network to its noisy counterpart is related to the graph, rather than subgraph, isomorphism problem, because both networks are of the same size, NA methods are designed to deal with the more general subgraph isomorphism problem, i.e., aligning networks that are of different sizes.

Because node2vec and struc2vec are based on random walks, and because random walks can be thought of as "sampled graphlets", we expect that node2vec and struc2vec will be faster than graphlets. However, we find that this is *not* the case – graphlets are an order of magnitude faster than both node2vec and struc2vec, even though the implementations of the latter two are parallelized, unlike the implementation of graphlets that we use. At the same time, graphlets overall yield more accurate alignments of the analyzed networks compared to both node2vec and struc2vec. Hence, our results imply that graphlets should be adopted from the computational biology domain to the social network domain.

2 METHODS

2.1 Measuring node similarities

Graphlets. Here, we provide more intuition behind graphlet automorphism orbits. Given a graphlet, nodes that are topologically symmetric to each other are a part of the same orbit. Consider a three node path (G_1 in Fig. 1). The two nodes on the end of the path

are symmetric to each other, so they are in the same orbit (colored black and labeled with a 1 in G_1). On the other hand, the node in the middle is only symmetric to itself, and thus in its own distinct orbit (colored white and labeled with a 2 in G_1). So, if node u in a network is on the end of a three node path, it "participates" in the first orbit. If it is in the middle of a three node path, it "participates" in the second orbit. These orbits are precomputed for all graphlets of up to five nodes. So, to form a node's GDV, for each orbit, one counts how many times that node participates in the orbit. Each position in the GDV corresponds to an orbit, so if we consider up to 4-node graphlets (which we do in this study), we have a feature vector of length 15, because 2-4-node graphlets have 15 orbits (Fig. 1). To compute the GDVs, we use the Orca tool [14].

Node2vec. Node2vec uses biased random walks (intuitively, these random walks follow the breadth-first search and depth-first search strategies) to learn the features of a node. In a random walk with some starting node u, in the first step there is an equal chance of visiting every neighbor of u. In the second step, starting from this neighbor of u, there is again an equal chance of visiting every neighbor. This continues until a specified number of steps is made. In a *biased* random walk, in each step some neighbors have a higher or lower chance of being visited.

In node2vec, given a node u, the algorithm performs a number of biased random walks of a certain length starting at u. Then, for every other node in the network, these walks are used to obtain the probability of the node u being close in distance to the other node. For example, if many random walks starting at u encounter some other node v within a few steps, then there is a high probability u is close to v. This also means the feature vector for u will be similar to that of v. For the formal approach of node2vec, see [10].

We use the default parameter values of the C++ implementation of node2vec provided in the SNAP GitHub; these values were empirically chosen in the node2vec study. While node2vec is sensitive to changes in some of these parameter values [10], it is not in the scope of this study to find the best performing parameter values for every network pair and alignment method. Actually, for this particular reason—the need to always determine the best-performing parameter values—the parameter sensitivity of node2vec can be seen as one of its drawbacks.

Struc2vec. Ribeiro *el al.* introduce struc2vec as another biased random walk-based method. The main difference between node2vec and struc2vec is that while in node2vec random walks occur on the original network, in struc2vec they occur on a modified version of the network where nodes that are close in distance in this network are structurally similar in the original network. Using the node2vec example above, if node u encounters some other node v within a few steps in many random walks, there is a high probability u is close, and therefore structurally similar, to v. Again, the feature vector of u will be similar to that of v. For the formal approach of struc2vec, see [26].

Ribeiro *el al.* argue that while methods like node2vec work well for node classification tasks, they tend to fail for structural equivalence tasks. In particular, given a node with a certain feature, neighbors of the node are likely to have the same feature. As such, nodes that are close in distance will more likely have similar feature vectors compared to nodes that are far in distance. So, structural equivalence will not necessarily be captured very well.

We use the default parameters provided in the struc2vec GitHub; these values were empirically chosen in the struc2vec study. Again, while struc2vec is sensitive to changes in some of these parameter values [26], it is not in the scope of this study to find the best performing parameter values for every network pair and alignment method.

Quantifying node similarity between networks. Given two nodes and their respective feature vectors, we calculate the cosine similarity between them, which we then normalize between 0 and 1 by adding the maximum possible value (1) and dividing by the range (2). While the inverse of Euclidean distance is also an option for a similarity measure, empirically we have found that cosine similarity typically gives better results. Note that for graphlets, we first perform principal component analysis (PCA), a standard dimension reduction technique, on the GDVs of all nodes from the networks being aligned, and then we compute cosine similarity between the PCA-reduced GDVs. We use PCA because we have found empirically that doing so almost always gives better alignment quality than not using PCA; of course, there could exist another dimensionality reduction technique that might yield even better alignment quality. We choose the first r principal components, where r is at least two and as small as possible such that the r components account for at least 90% of the variation in the data. For the node2vec and struc2vec methods, we do not employ any dimensionality reduction technique, as the dimension of the embedding is a method parameter.

2.2 Alignment strategies

WAVE. WAVE takes as input two networks and similarities between all pairs of nodes from the two networks. Then, it uses a "seed-and-extend" algorithm to align the networks. First two highly similar nodes are aligned, i.e., seeded. Then, the seed's neighboring nodes that are similar are aligned, the seed's neighbor's neighbors that are similar are aligned, and so on. The extension step continues until all nodes in the smaller of the two compared networks are aligned (formally, until a one-to-one node mapping between the two networks is produced).

SANA. SANA also takes as input two networks and similarities between all pairs of nodes from the two networks, like WAVE. However, SANA uses a search algorithm, specifically simulated annealing, to find an alignment. That is, instead of aligning networks node by node as WAVE does, SANA explores the space of possible alignments and find the highest scoring one with respect to the objective function. We set the following parameters for SANA: `s3` to 1, `esim` to 1, `simFile` to the name of the node similarity file, and `simFormat` to 1 (this tells SANA to read the similarity file such that each line has 3 columns: node1, node2, and the similarity between them). SANA also has a running time parameter; for the smaller synthetic networks (discussed below), we set `t` to 5 minutes. For the larger biological networks (discussed below) we increase `t` to 60 minutes since SANA requires more time to find a good alignment (which we have verified empirically in our evaluation).

3 RESULTS AND DISCUSSION

3.1 Evaluation

We study the three embedding (i.e., node similarity) methods – graphlets, node2vec, and struc2vec (Section 2.1). We run each embedding method under each of the two alignment strategies – WAVE and SANA (Section 2.2). We evaluate the three embedding methods under the same alignment strategy by aligning two types of networks – synthetic and real-world biological (PPI) networks, to their noisy counterparts, as follows.

Synthetic networks. We form synthetic networks using two random graph generators, namely: 1) geometric random graphs (GEO) and 2) scale-free networks (SF). The two models have distinct network topologies [21], which enables us to test the robustness of our results to the choice of random graph model. We set both networks to the same size of 1,000 nodes and 6,000 edges.

PPI networks. We consider two different types of PPIs (i.e., edges): only affinity capture coupled to mass spectrometry (APMS), and only two-hybrid (Y2H). Sizes of the two PPI networks are shown in Table 1. The two edge types reflect biological experiments used to detect the PPIs. Intuitively, APMS experiments will result in networks that will have more clique-like structures, and Y2H experiments will result in networks that will have more star-like structures. That is, the two PPI networks have different topologies, which allows us to test the robustness of our results to the choice of PPI type.

Network	# of nodes	# of edges
APMS	11,450	92,257
Y2H	10,317	41,925

Table 1: Sizes of the two considered PPI networks.

Creating noisy counterparts of a synthetic or PPI network. A noisy counterpart is the original network with x% of its edges rewired, where we vary x to be 0, 10, 25, 50, 75, and 100. Since only edges are changed between the original and noisy network, we know which nodes should be mapped to which. We can use this true node mapping to accurately evaluate our methods; a good method should have high node correctness, which is the percentage of node pairs from the given alignment that are present in the true node mapping. Other evaluation schemes, such as the fraction of correctly mapped nodes out of nodes that have no rewired edges, could be interesting to explore, but this is out of the scope of this study. We form five rewired network instances at each noise level to account for the randomness of the edge rewiring process, though more instances can be used to increase the confidence in the results. Then, we average the alignment quality over the multiple runs corresponding to the multiple instances.

3.2 Method comparison in terms of accuracy

We expect that a good method (i.e., a combination of an embedding method and an alignment strategy) should have high alignment quality at low noise levels, and low alignment quality at high noise levels, with a general decreasing trend as noise increases. This is because at low noise levels, we are aligning two networks with similar topologies compared to each other, while at high noise levels, we are aligning two networks with almost random topologies compared to each other. We also expect that a good method should be robust to noise. That is, we should see a slower decrease in alignment quality as noise increases compared to other methods. We find that graphlets fit all of these criteria (Figs. 3 and 4), unlike node2vec or struc2vec.

Specifically, we first perform a summary analysis that aims to rank the three embedding approaches against each other. Considering up to 50% noise (which results in 4 networks × 4 noise levels × 2 alignment strategies = 32 evaluation tests), we count how many times out of the total of 32 evaluation tests a given embedding method is the best (rank 1), second best (rank 2), or the third best, i.e., worst (rank 3) in terms of alignment quality. Note that in this ranking analysis we omit the largest noise levels of 75% and 100% because for most of the methods and networks, alignment quality is tied at near 0 values, as expected for such random-like noise levels.

In the ranking analysis, we find that graphlets are the best approach (have rank 1) most of the time – in 90.63% of all evaluation tests (Fig. 3). We do observe some ties between the methods. In particular, all three methods are tied with each other in 21.88% of all cases, and graphlets are tied with node2vec (but not with struc2vec) in 6.25% of additional cases. This means that graphlets are superior (without ties) to both node2vec and struc2vec in 62.5% of all cases, and are tied to at least one of node2vec or struc2vec in additional 21.88% + 6.25% = 28.13% of all cases. Node2vec is superior to the other two methods in only 6.25% of all cases. Struc2vec is superior to the other two methods in only 3.125% of all cases. This analysis questions the usefulness of node2vec and struc2vec, as they improve upon graphlets in only two and one out of the 32 cases, respectively.

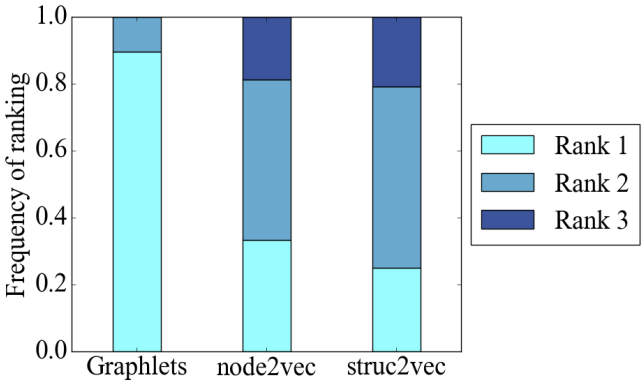

Figure 3: Summarized results regarding the effect of the embedding method (x-axis) on alignment quality over 32 evaluation tests (4 networks × 4 noise levels × 2 alignment strategies). For each test, we compare the different methods and rank them from the best (rank 1) to the worst (rank 3). The figure shows the percentage (frequency) of all evaluation tests in which the given method has the given rank. Note that in the figure, some rank ties exist (see the text for details).

Second, we consider the detailed alignment quality versus noise results for each embedding method, network, and alignment strategy. This detailed analysis again confirms the superiority of graphlets (Fig. 4). Specifically, at low noise levels, graphlets have high alignment quality, and at high noise levels, graphlets have low alignment quality, with a decreasing trend, as expected. Contrast this with node2vec's results, where at some higher noise levels the alignment quality is actually better than at lower noise levels, which is a trend that should not happen, as the networks being aligned are more similar and should thus yield higher alignment quality at lower than at higher noise levels. Regarding struc2vec, while this method overall shows the expected trends, just as graphlets, it is less accurate than graphlets. Also, of all approaches, we find that graphlets are the most robust to noise, showing a slower decrease in alignment quality as noise increases compared to the other two methods.

3.3 Method comparison in terms of running time

We also analyze running times of the different embedding methods. All methods are run on a 64-core AMD Opteron 6376 machine with 500 GB of RAM. We record both real times (the actual clock time elapsed) and CPU times (the total amount of time the core(s) spend executing) for all methods. The more cores are used, the lower the real time is expected to be. Hence, it is more fair to compare the different methods' CPU times, which essentially reflect how long the given method would take if a single core was used. This is especially true because node2vec uses 10 cores, struc2vec uses four cores, and the implementation of graphlets that we consider uses only one core. Again, if one wishes to ignore implementation-specific (dis)advantages of the given method, including (lack of) parallelization, it is more fair to compare the different methods' CPU running times. On the other hand, if one wishes to give each method/implementation the best-case advantage, then the methods' real times should be compared.

It is important to note that the running time of counting graphlets depends on the largest considered graphlet size (in this case, we use up to 4-node graphlets), and the running times of computing node2vec and struc2vec depend on parameters such as the length of the random walks and the number of random walks per node (in this case, we use the default parameters of each method). Since the different methods depend on different parameters whose considered values are not necessarily fairly comparable, the different methods' running times might not be fairly comparable either. Nonetheless, since graphlets are superior to the other two methods in terms of accuracy under the considered parameter values, we are interested in how the different methods' running times compare under the same parameter values.

We expect computing node2vec and struc2vec to be faster than counting graphlets, because the former two are based on random walks, which can be thought of as "sampled graphlets". However, we find that counting graphlets, i.e., obtaining nodes' GDVs, is faster than computing the nodes' node2vec or struc2vec feature vectors. This holds for all four analyzed networks, and independent on whether we consider real or CPU running times (Table 2). Also, we note that struc2vec is slower than node2vec. One possible explanation for this is that struc2vec has an extra step compared to node2vec – that of creating a modified version of the original network when performing random walks.

The superiority of graphlets in terms of running time is most likely due to the graphlet implementation that we use called Orca, which leverages combinatorial relationships between the different graphlet orbits. That is, by knowing the counts of some graphlets (i.e., orbits), Orca can infer the counts of the other graphlets through mathematical equations, rather than having to actually count these graphlet occurrences by traversing the network structure. Consequently, Orca significantly speeds up computation of graphlet counts compared to the naive implementation of graphlet counting that would explicitly search the network structure for the occurrence of every graphlet, such as that implemented in the GraphCrunch tool [21].

Table 2: Running times of the embedding methods on each network, in seconds. In the table, real running time refers to the actual clock time elapsed, while CPU running time refers to the amount of time the core(s) spend executing.

	GEO	SF	APMS	Y2H
graphlets-real	0.03	0.03	1.08	0.26
graphlets-CPU	0.02	0.02	1.02	0.22
node2vec-real	9.75	13.38	105.56	122.91
node2vec-CPU	563.40	777.73	6426.68	7504.04
struc2vec-real	75.37	77.09	2874.24	1942.90
struc2vec-CPU	268.03	282.43	10998.21	7333.74

4 CONCLUSION

In summary, we compare the three network embedding methods – graphlets, node2vec, and struc2vec, in the task of network alignment. Specifically, for a given embedding method, we use the features generated by it to calculate node similarities. Then, we use the node similarity information in two existing network alignment strategies, WAVE and SANA. We fairly evaluate the different embedding methods under the same alignment strategy, and we do so by aligning synthetic and PPI networks to their noisy counterparts. We find that graphlets generally outperform the other embedding methods in terms of alignment quality. Importantly, node2vec and struc2vec improve upon graphlets in under 6.25% of all evaluation tests, which questions their usefuleness compared to graphlets, at least in the considered task of network alignment. Furthermore, not only do we find graphlet-based alignments to be the most accurate, but we also find that counting graphlets is faster than computing node2vec and struc2vec. We use the Orca implementation for graphlet counting. There also exist more recent implementations than Orca that aim to speed up graphlet counting, e.g., by parallelizing the counting or estimating counts through sampling, which consequently makes graphlet counting possible in very large networks with millions of nodes/edges [1, 24, 35]. These implementations could further speed up graphlet counting.

Evaluating how graphlets perform against node2vec and struc2vec in tasks other than that of network alignment, such as node classification, network clustering, or link prediction, is the subject of future work.

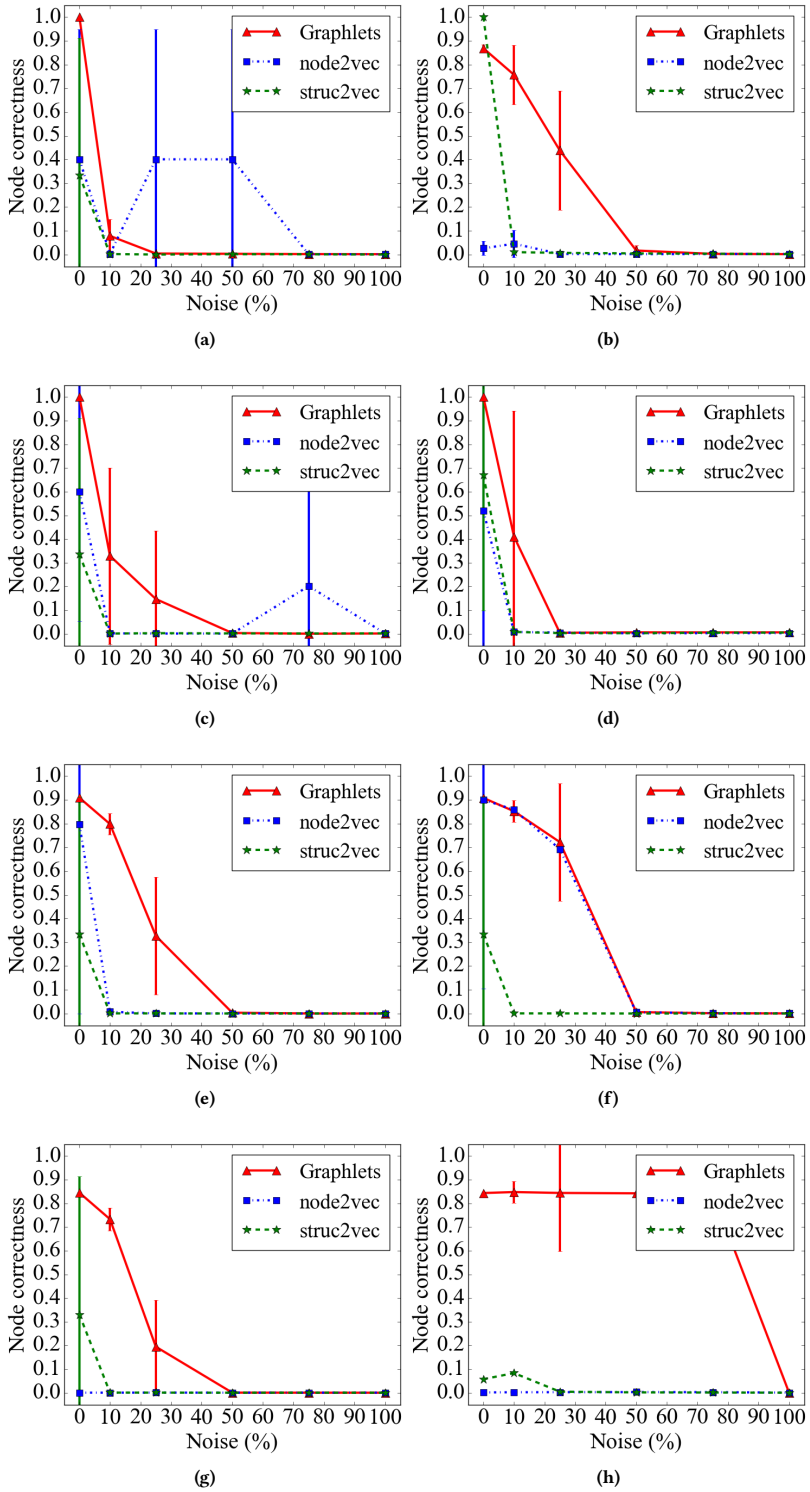

Figure 4: Detailed alignment quality results regarding the effect of the embedding method on alignment quality as a function of noise for (a,b) geometric synthetic networks, (c,d) scale-free synthetic networks, (e,f) the APMS PPI real-world network, and (g,h) the Y2H PPI real-world network, under (a,c,e,g) WAVE alignment strategy and (b,d,f,h) SANA alignment strategy.

Importantly, while graphlets were originally introduced in the context of static and homogeneous networks [20, 25], given recent availability of dynamic (temporal, evolving) or heterogeneous (multi-node or multi-edge type) networks, graphlets have been extended into their dynamic [15] or heterogeneous [11] counterparts. Then, dynamic or heterogenous graphlets have been used in newly defined network science tasks of aligning dynamic (rather than traditionally static) networks [31, 33], clustering a dynamic network based on topological similarity rather than the traditionally considered notion of network denseness [4], and aligning heterogeneous (rather than traditionally homogeneous) networks [11].

Therefore, we believe that graphlets will continue to make their mark in the field of network science, hopefully across its many interdisciplinary domains, including biological, social, technological, information, transportation, infrastructure, ecological, climate, and other networks.

ACKNOWLEDGEMENTS

This work was funded by Air Force Office of Scientific Research (AFOSR) Young Investigator Research Program (YIP) under award number FA9550-16-1-0147, and National Science Foundation (NSF) Faculty Early Career Development Program (CAREER) under award number CCF-1452795.

REFERENCES

[1] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *Data Mining (ICDM), 2015 IEEE International Conference on*. IEEE, 1–10.
[2] Albert-László Barabási. 2016. *Network science*. Cambridge University Press.
[3] Stephen A Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM Symposium on Theory of Computing*. ACM, 151–158.
[4] Joseph Crawford and T Milenković. 2018. ClueNet: Clustering a temporal network based on topological similarity rather than denseness. *PLOS ONE, in press* (2018).
[5] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. 2017. A Survey on Network Embedding. *arXiv preprint arXiv:1711.08752* (2017).
[6] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 135–144.
[7] Fazle E Faisal, Lei Meng, Joseph Crawford, and Tijana Milenković. 2015. The post-genomic era of biological network alignment. *EURASIP Journal on Bioinformatics and Systems Biology* 2015, 1 (2015), 3.
[8] Fazle E Faisal and Tijana Milenković. 2014. Dynamic networks reveal key players in aging. *Bioinformatics* 30, 12 (2014), 1721–1729.
[9] Fazle Elahi Faisal, Han Zhao, and Tijana Milenković. 2015. Global network alignment in the context of aging. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 12, 1 (2015), 40–52.
[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 855–864.
[11] Shawn Gu, John Johnson, Fazle E Faisal, and Tijana Milenković. 2017. From homogeneous to heterogeneous network alignment. *arXiv preprint arXiv:1704.01221* (2017).
[12] Pietro Hiram Guzzi and Tijana Milenković. 2017. Survey of local and global biological network alignment: the need to reconcile the two sides of the same coin. *Briefings in Bioinformatics* (2017). https://doi.org/10.1093/bib/bbw132
[13] Steve Harenberg, Gonzalo Bello, L Gjeltema, Stephen Ranshous, Jitendra Harlalka, Ramona Seay, Kanchana Padmanabhan, and Nagiza Samatova. 2014. Community detection in large-scale networks: a survey and empirical evaluation. *Wiley Interdisciplinary Reviews: Computational Statistics* 6, 6 (2014), 426–439.
[14] Tomaž Hočevar and Janez Demšar. 2014. A combinatorial approach to graphlet counting. *Bioinformatics* 30, 4 (2014), 559–565.
[15] Yuriy Hulovatyy, Huili Chen, and Tijana Milenković. 2015. Exploring the structure and function of temporal networks with dynamic graphlets. *Bioinformatics* 31, 12 (2015), i171–i180.
[16] Yuriy Hulovatyy, Ryan W Solava, and Tijana Milenković. 2014. Revealing missing parts of the interactome via link prediction. *PLOS ONE* 9, 3 (2014), e90073.
[17] Fragkiskos D Malliaros and Michalis Vazirgiannis. 2013. Clustering and community detection in directed networks: A survey. *Physics Reports* 533, 4 (2013), 95–142.
[18] Nil Mamano and Wayne B. Hayes. 2017. SANA: simulated annealing far outperforms many other search algorithms for biological network alignment. *Bioinformatics* 33, 14 (2017), 2156–2164.
[19] Lei Meng, Aaron Striegel, and Tijana Milenković. 2016. Local versus global biological network alignment. *Bioinformatics* 32, 20 (2016), 3155–3164.
[20] Tijana Milenkovć and Nataša Pržulj. 2008. Uncovering Biological Network Function via Graphlet Degree Signatures. *Cancer Informatics* 6 (2008).
[21] Tijana Milenković, Jason Lai, and Nataša Pržulj. 2008. GraphCrunch: a tool for large network analyses. *BMC bioinformatics* 9, 1 (2008), 70.
[22] Arvind Narayanan, Elaine Shi, and Benjamin IP Rubinstein. 2011. Link prediction by de-anonymization: How we won the kaggle social network challenge. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE, 1825–1834.
[23] Mark Newman. 2010. *Networks: an introduction*. Oxford University Press.
[24] Mark Ortmann and Ulrik Brandes. 2016. Quad census computation: Simple, efficient, and orbit-aware. In *International Conference and School on Network Science*. Springer, 1–13.
[25] Natasa Pržulj, Derek G Corneil, and Igor Jurisica. 2004. Modeling interactome: scale-free or geometric? *Bioinformatics* 20, 18 (2004), 3508–3515.
[26] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 385–394.
[27] Omkar Singh, Kunal Sawariya, and Polamarasetty Aparoy. 2014. Graphlet signature-based scoring method to estimate protein–ligand binding affinity. *Royal Society Open Science* 1, 4 (2014), 140306.
[28] Ryan W Solava, Ryan P Michaels, and Tijana Milenković. 2012. Graphlet-based edge clustering reveals pathogen-interacting proteins. *Bioinformatics* 28, 18 (2012), i480–i486.
[29] Yihan Sun, Joseph Crawford, Jie Tang, and Tijana Milenković. 2015. Simultaneous Optimization of both Node and Edge Conservation in Network Alignment via WAVE. *Lecture Notes in Computer Science Algorithms in Bioinformatics* (2015), 16–39.
[30] Yihan Sun, Joseph Crawford, Jie Tang, and Tijana Milenković. 2015. Simultaneous Optimization of both Node and Edge Conservation in Network Alignment via WAVE. *Lecture Notes in Computer Science Algorithms in Bioinformatics* (2015), 16–39.
[31] Vipin Vijayan, Dominic Critchlow, and Tijana Milenković. 2017. Alignment of dynamic networks. *Bioinformatics* 33, 14 (2017), i180–i189.
[32] Vipin Vijayan, Eric Krebs, Lei Meng, and Tijana Milenković. 2017. Pairwise versus multiple network alignment. *arXiv preprint arXiv:1709.04564* (2017).
[33] Vipin Vijayan and Tijana Milenković. 2017. Aligning dynamic networks with DynaWAVE. *Bioinformatics* (2017). https://doi.org/10.1093/bioinformatics/btx841
[34] Vipin Vijayan and Tijana Milenković. 2017. Multiple network alignment via multiMAGNA++. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (2017). https://doi.org/10.1109/TCBB.2017.2740381
[35] Pinghui Wang, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John Lui, Don Towsley, Junzhou Zhao, Jing Tao, and Xiaohong Guan. 2016. A fast sampling method of exploring graphlet degrees of large directed and undirected graphs. *arXiv preprint arXiv:1604.08691* (2016).
[36] Xiao-Dong Wang, Jia-Liang Huang, Lun Yang, Dong-Qing Wei, Ying-Xin Qi, and Zong-Lai Jiang. 2014. Identification of human disease genes from interactome network using graphlet interaction. *PLOS ONE* 9, 1 (2014), e86142.
[37] Yang Yang, Ryan N Lichtenwalter, and Nitesh V Chawla. 2015. Evaluating link prediction methods. *Knowledge and Information Systems* 45, 3 (2015), 751–782.
[38] Ömer Nebil Yaveroğlu, Tijana Milenković, and Nataša Pržulj. 2015. Proper evaluation of alignment-free network comparison methods. *Bioinformatics* 31, 16 (2015), 2697–2704.
[39] Yutao Zhang, Jie Tang, Zhilin Yang, Jian Pei, and Philip S Yu. 2015. Cosnet: Connecting heterogeneous social networks with local and global consistency. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1485–1494.
[40] Marinka Zitnik and Jure Leskovec. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* 33, 14 (2017), i190–i198.