Enhancing the Random Walk Parametrization in the *node2vec* Graph Embedding Algorithm

Filip Ryzner, El Ghali Zerhouni and Makram Chahine

I. Introduction

We explore a possibility for enhancing the representation power of a popular graph embedding algorithm, *node2vec* [1], [2], which uses a biased random walks to learn direct encoding of nodes. The *node2vec* algorithmic framework learns a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes, thus nodes that co-occur on short random walks over the graph should have similar embeddings. However, the in-out hyperparameter p and return hyperparameter q defining the biased random walk behaviour are constant over all nodes of the graph. Moreover, no prior information about the graph structure is leveraged to tune the transition probability hyperparameters p and q. Indeed, in the original work, the method used on each dataset is 10-fold cross-validation on 10% of the labeled data with a 5×5 grid of possible values. Our conjecture is that there is room for performance and hyperparameter selection improvement while preserving the linear time computation of the method. All our tests are run on the link prediction task and

The novel contributions presented in this work are:

i) Design of a linear time expansion of *node2vec* taking into account neighborhood structure through first and second degree centrality,

ii) Fine tuning of the designed approach to be deployable out of the box on any graph,

iii) Optimization of the original *node2vec* hyperparameters on baseline graphs, thus providing a best-case score to compare our method against,

iv) Validation of the proposed method against a similar approach with random p, q at each node and against the optimized original *node2vec* on baseline graphs.

II. Related work

Multiple variants of the algorithm have subsequently been developed in broader literature, none to our knowledge focuses on the problem of computing informed transition probability parameters p, q. Most random walk graph embedding algorithms directly modify the *DeepWalk* algorithm introduced by [3]. The *node2vec* algorithm, is also based on *DeepWalk*, with the significant difference being that *DeepWalk* uses unbiased random walk of length T, while *node2vec* features a walk bias variable (parameterized by p and q). Elsewhere, the [4] introduced a modification to *DeepWalk* utilizing attention and back-propagation to determine the optimal length of the random walk, which in [3] is a fixed parameter. Moreover, in [5], the concept of the random walk based embeddings is expanded from homogeneous graphs to heterogeneous graphs and setting the state-of-the-art for various tasks on heterogeneous graphs. Finally, in [6], the random walk embedding algorithm is modified to identify nodes that play a similar role, thus better extracting graph structural information compared to other algorithms.

III. Technical approach

Our proposed method approaches the problem by treating each node differently and adapting the transition probabilities to adjacent nodes given information about the neighborhood. Indeed, we design a mapping taking first and second order node degree centrality as input and producing the random walk parameters p, q at the given node as output. The *node2vec* random walks are thus generated with customized biases.

A. Problem formulation

Given a graph G with an edge set E and c_i as the i-th step of walk over the graph, we define π_{vx} as the unnormalized transition probability from vertex v to vertex x, and Z as the normalisation constant which gives us:

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v,x) \in E \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

These random walks are characterized by the type of search they are enforcing. A Depth First Search (DFS) prioritizing random walk results in arrival to other nodes that are far from the node of origin, while a Breadth First Search (BFS) enforces exploration of the more immediate environment of each node.

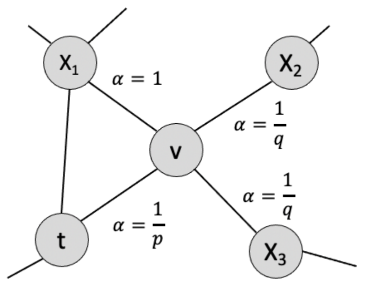

Fig. 1. *node2vec* random walk with the parameters p and q adjusting DFS and BFS. Example from Grover2016.

Specifically, *node2vec* defines a second order biased random walk with two parameters p and q which determine the probability of returning to the previous vertex and deeper exploration respectively. As illustrated in Figure 1, given the walk that just went through the edge (t,v) and is now at node v, the transition probability between v and x is $\pi_{vx} = \alpha_{pq}(t,x)$ such that:

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \quad (2)$$

The current *node2vec* approach relies on a cross validation on a randomly selected 10% of the graph to define the optimal p and q values, which are then held constant for all vertices in the graph. We consider this to be sub-optimal as for nodes that are isolated (less connected to others) in the graph, a DFS approach would better characterize their environment, while for the central nodes of the graph (well connected to others), a BFS approach seems preferable to. In Figure 2, we observe that if the cross validation has been applied on node x_7 to x_{10}, then DFS would have been favored, but this won't be convenient for node x_3 which is highly central in the graph, thus a BFS would better map its environment.

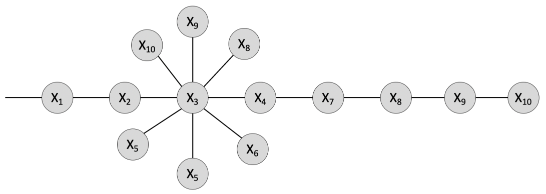

Fig. 2. Example of a graph where the current *node2vec* strategy is sub-optimal.

B. Centrality leverage

More formally, we leverage the notion of degree centrality, that computes the degree of each node in the graph. Other centrality metrics exist in the graph but we particularly choose the degree centrality as it is computed linearly in the number of nodes in the graph. Hence, for each node i, we define its degree centrality $d_1(i)$. The more central this node is, the more a BFS approach should be favored. Let d_1^{max} be the maximum degree centrality. We could adjust the parameter p_i for each node of origin i such as:

$$p_i \propto \frac{d_1^{max}}{d_1(i)}$$

On the other hand, a DFS approach could be favored when the nodes that the origin node is connected to have high centrality values. In the sense, that going one step forward in the process would enable the search to discover a richer environment in the graph. Figure 3 shows an example where node x_7 have highly connected neighbors and we would get more information from DFS approach.

We then define for node i, the second degree centrality as the mean of the centralities of the nodes in its direct environment:

$$d_2(i) = \frac{\sum_{j,(i,j)\in E} d_1(j)}{d_1(i)}$$

Let d_2^{max} be the maximum second degree centrality. Hence, we could adjust the parameter q_i for each node of origin i

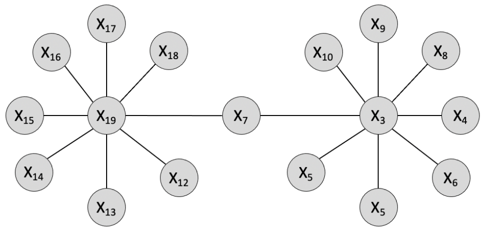

Fig. 3. Example of a graph where favoring DFS in node 7 provides more information.

such as:

$$q_i \propto \frac{d_2^{max}}{d_2(i)}$$

Computing this centrality measure is linear with the number of nodes in the graph and is then scalable. Moreover, the mapping $\phi : (d_1, d_2) \rightarrow (p, q)$ we propose is, for constants to be tuned α, β,

$$\phi(d_1, d_2) = (\alpha\frac{d_1^{max}}{d_1(i)}, \beta\frac{d_2^{max}}{d_2(i)}) \quad (3)$$

Constants α, β are tuned over a variety of specific graph architectures, then kept constant for all datasets tested. This is a central point to this method, otherwise we would just be replacing the tuning of p, q by that of α, β on each and every graph. The tuning method is extensively discussed in section V-B.

IV. Methodology

A. Algorithms

Three models are tested against one another for a given number of random walks per node and fixed length: A.

i) Original *node2vec* : constant p, q for all nodes
ii) Our method : p_i, q_i for node i given by ϕ as per (3)
iii) Randomized selection : p_i, q_i for node i randomly sampled from uniform distribution

B. Hyperparametrization

In the original [1] paper, state-of-the-art results are achieved for the link prediction task with 50% of edges removed from the networks. However, no values are given for the hyperparameters used to reach such performances. Given the sizes of the graphs and the number of hyperparameters one would have to tune, reproducing the performance levels from the original paper would be a very time consuming matter. Furthermore, the ratio of utility to time of such an endeavor is infinitesimal, given we seek to expand on the random walk transition probabilities influenced only by the hyperparameters p and q.

One example provided in [7] does provide *node2vec* hyperparameter values and a performance score on the Cora citation dataset that we use as a reference sanity check for our algorithms. In fact, the hyperparameters used are the default values coded in the original paper (that do not lead to the high performances presented in the paper). In more detail, the dimension of the embedding is set at 128, the number of random walks initiated at each node is 10 and the length of each random walk is 80. We retain these parameters throughout our study since we do not need to reach optimal performance in the absolute sense but only to compare various strategies for transition probability computation. To also be able to compare to the reference case, we only remove 10% of the original graph's edges.

C. Binary operators

Four binary operators are introduced for learning edge features, namely the Average, Hadamard, Weighted-L1 and Weighted-L2 operators. In the original work it is empirically established that the Hadamard binary operator is generally superior. For simplicity and as we are looking for relative performance changes, we solely use the Hadamard operator in all our experiments.

D. Dataset selection

The Cora dataset is preferred for tuning parameters for multiple reasons. It is smaller than other baseline graph datasets allowing for faster hyperparameter tuning. Also as discussed above, this dataset allows us to have a complete reference to work with. It is thus used to tune the α and β parameters empirically.

We also evaluate the performance of the suggested embedding solutions on the CiteSeer dataset. We optimize the original *node2vec* random walk biases and compare to the tuned modified solution and the randomized bias approach. This provides us a completely independent test case to evaluate our proposed solution.

Remark 4.1: We mention that testing on larger datasets requires extensive hyperparameter tuning as well as very long computation time to reach better than random guess performance. This is a limiting factor in this study.

E. Scoring

We retain the popular multi-class classification score, the AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes of edges (positive/negative).

For coherence we compare all algorithms trained on the same train subset of a given dataset and test them on the same test subset.

F. Stochastic nature

Given the stochastic nature of the random walks in the algorithms, repeatability is far from respected. Indeed, multiple runs of the same algorithm on the same test set can score differently. The deltas can be large enough to mask any performance change that might emerge from hyperparameter tuning or algorithm modification.

In order to mitigate this effect, we average the score for each algorithm and hyperparameter choice over 10 link prediction runs (the standard deviation over 10 runs is on average around 0.02, always smaller than 0.04).

G. α and β tuning

The tuning of the proposed mapping coefficients α and β given in (3) is supposed to be universal. In other terms, we wish to fix a unique pair of values for these coefficients that would capture how random walks link first and second degree centrality of a node to information maximization over its neighbourhood.

One method used is empirical fine-tuning on the Cora dataset. We start by a coarse grid search to determine the suitable order of magnitude of each coefficient then a finer search around interesting values.

V. Empirical Results

We present the quantitative results of the study in this section which is organized as follows. First, we show that our implementation does not come with computation time cost, then we exhibit the numerical tuning of the mapping coefficients α, β, next we tune the original *node2vec* on each of the datasets and finally evaluate the three algorithms on the aforementioned datasets.

A. Time cost of bias customization

The proposed method for computing specific random walk biases at each node is evidently more computationally expensive than the original algorithm as it requires additional operations to be executed. However, the computation time difference between the two approaches is negligible. Moreover, the transition probability computation is not necessarily the longest process in the algorithm. It appears that generating the random walks consumes an order of magnitude more time than computing the transition probabilties (even parallelized on 16 GPUs). As the size of the graphs increase, transition probability computation does require more time, however remains in the same order of the unaltered *node2vec*, and smaller than the time to generate the random walks on the graph. These results are for 10 random walks of length 80 at each node, which is not enough to learn rich enough embeddings on the larger graphs such as the Facebook dataset (the random walk generation time with a suitable selection of hyperparameters would take much longer). We can thus reaffirm that our method comes at no practical additional time cost. These observations can be inferred from the computation durations of the different processes presented in Table I.

Dataset	n2v TP (s)	ours TP (s)	RW gen (s)
Cora	0.20	0.22	1.72
CiteSeer	0.15	0.15	1.23
Facebook	19.5	25.1	27.8

TABLE I
Table containing the average computation times of transition probabilities for the original *node2vec* and our solution, and of random walks generation (solution independent)

B. α and β tuning

In the original *node2vec* paper a sensitivity analysis was conducted and the conclusions on the values of p and q were the following: *"The performance of node2vec improves as the in-out parameter p and the return parameter q decrease [...] While a low q encourages outward exploration, it is balanced by a low p which ensures that the walk does not go too far from the start node."* On the Cora set which we use for tuning or node-specific p and q values, this tendency seems to hold with nuance. We first run a very coarse 11×11 grid search over α and β in logarithmic scale with values ranging between 0.01 and 100. The AUC-ROC scores standard deviation for each hyperparameter combination over the 10 runs is ~ 0.02, and the average scores obtained are summarized in Figure 4.

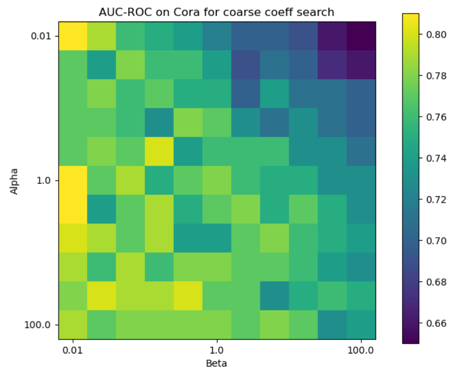

Fig. 4. Coarse logarithmic AUC-ROC tuning of α and β on the Cora dataset

We set the context for interpreting these scores by recalling that for each node the customized p, q values are given by (3), hence we have the direct correspondence that p, q and α, β respectively vary proportionately. Also we signal that the designed centrality terms by which we multiply α, β both numerically range in order of magnitude between 1 and 10^2. We now observe that there is a corner corresponding to small α and large β that heavily penalizes performance. In fact, increasing α for big β seems to mitigate the problem. This seems to contradict the aforementioned conjecture in the initial work, but the values considered for p, q were at least 2 orders of magnitude smaller than the largest α, β pairs. When considering the region $\beta \leq 1$ we notice that increasing α does not have the same effect. In fact, the score differences become much more diluted.

Thus, we set up a finer 11×11 grid search over α and β in logarithmic scale with values ranging between 10^{-3} and 1. The findings are presented in Figure 5.

We notice similarities in the patters of scoring between the coarse and finer grid searches: having $\beta > \alpha$ generally yields bad results. We observe a neighborhood of satisfactory values in the bottom-left center of the grid. These values are centered around $\alpha = 0.1, \beta = 0.01$. These are the values we select to uniquely define the mapping ϕ at each node.

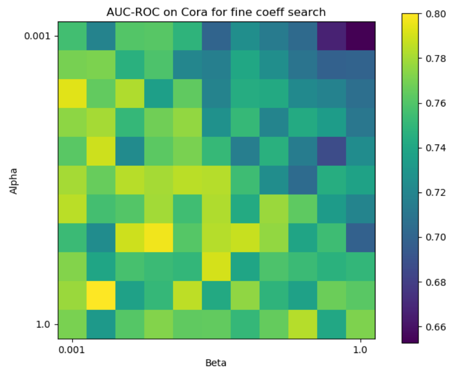

Fig. 5. Fine logarithmic AUC-ROC tuning of α and β on the Cora dataset

C. Original node2vec on baselines

We tune the original algorithm p, q parameters (with the other hyperparameters fixed to the values discussed earlier) on the 3 datasets we test on. We do a 5×5 grid search over the values suggested in the original paper, i.e. $[0.25, 0.50, 1, 2, 4]$. For Cora we select $p = 2, q = 0.5$ and for CiteSeer we select $p = 2, q = 0.25$.

D. Performance comparison results

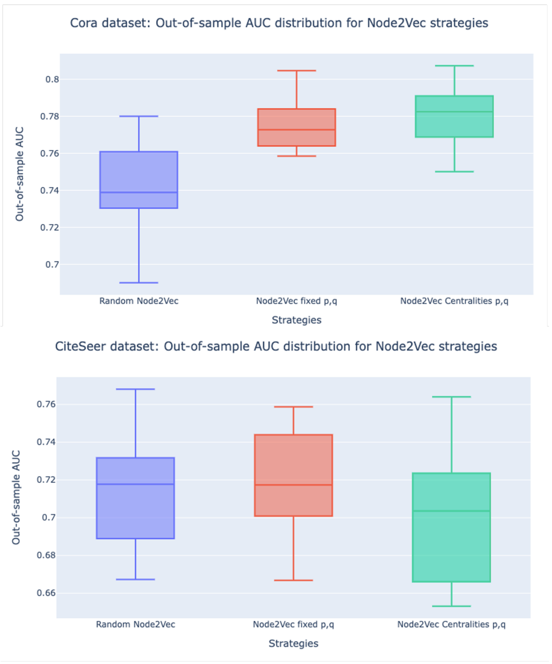

Fig. 6. Comparison of the out-of-sample AUC performance on the Cora and CiteSeer datasets of the Node2Vec strategies with random choice of p, q, constant p, q after cross-validation on the graph, node specific p, q depending on their centrality measures. Distributions are obtained after running each strategy 20 times.

We compare the three Node2Vec strategies: (a) with p, q selected at random at each node from a uniform $[0, 1]$ distribution ; (b) constant p, q after cross-validation on the entire graph ; (c) node specific p, q depending on their centrality

measures and cross-validated α and β, on the Cora and CiteSeer datasets. Given the randomness of the processes, we run each strategy 20 times on each data-sets, which results in the box distributions in Figure 6.
The results show that our strategy outperforms the classic Node2Vec and random strategies over the Cora dataset with a median out-of-sample AUC of 78.2% (25th quantile (q1): 76.9%, 75th quantile (q3): 79.1%) compared to a median of 77.2% (q1: 76.4%, q3: 78.4%) for the classic Node2Vec, and 73.9% (q1: 73.1%, q3: 76.1%) for the randomly generated p and q values. However, this was not the case of the CiteSeer datasets where the classic and random strategies has similar median performances, with respectively 71.7% (q1: 68.9%, q3: 73.1%) and 71.7% (q1: 70.1%, q3: 74.9%) out-of-sample AUC, while the centrality based centrality 1.4% less than the previous ones with 70.3% (q1: 66.6%, q3: 72.4%). Possible explanations of these mixed results could be that the centrality approach has an edge mainly on networks having having structural patterns that could be captured by the 1^{st} and 2^{nd} degree centrality we suggest. However, for pattern-less graphs where even random p and q values perform as well as calibrated ones, the centrality adjustment don't seem to provide additional value. Future research direction could go beyond the examples 2 and 3, and focus on automatically detecting the network patterns, such as the degree distribution, number and size of the clusters, sparsity of the adjacency matrix, where a centrality adjustments have a potential value.

VI. Conclusion

In conclusion, we have proposed a modification to the node2vec algorithm, which specifies random walk bias parameters p, q for each graph node individually through 1st and 2nd order node centralities, thus defines inidividual transition probabilities for each node. We note that due to high computation cost and extensive hyperparameter tuning required for relevant training on large datasets (more than 10k nodes), we evaluate our method on the Cora and CiteSeer datasets (which are smaller in size, but still commonly utilized for embedding comparisons and edge prediction).

We have shown that the increase in pre-processing time cost related to calculating biased random walk parameters for each node individually is not the limiting bottleneck when compared to the pre-processing times of the standard node2vec algorithm. Moreover, we explained that this is because most of the cost in pre-processing actually comes from the calculation of random walks rather than individual transition probabilities for nodes.

Furthermore, after tuning the α and β parameters, we use our modified node2vec algorithm for edge prediction on Cora and CitySeer datasets and comparing the performance with two benchmark algorithms, the first one being the node2vec with grid-search finetuned p, q and the second one being node2vec with a random choice of p, q. We have shown that while our method out-performed both benchmarks in terms of the AUC distribution (repeated trials due to randomness of the walk) on the Cora dataset, it did not do that well compared to the other two methods on the citeseer dataset. An important distinction for these two datasets was that the node2vec with random p, q was out-performed by the node2vec as well as our modified node2vec on the Cora dataset, yet on the citeseer dataset it produced to some extent similar performance to the standard node2vec and both out-performed our approach. We believe that the reason for this difference in performance comes from the structural difference between the two datasets. It seems that as our approach bases the finetuning of p, q of individual nodes on their 1st and 2nd order centralities, it offers performance edge on graphs, which have stronger structural patterns such as the Cora dataset. On contrary, on less structured graphs, our addition seems to actually hurt the overall performance in the edge prediction task.

Therefore, for future research we suggest to investigate the structure of the graph before deciding on which approach to utilize. Future research direction could focus on automatically detecting the network patterns and determining whether our centrality adjustments offers a potential value addition to the approach *node2vec* approach when used for edge prediction.

VII. Team Contributions

All produced submissions were result of extensive team collaboration. We held regular face-to-face team meetings where all key theories were discussed and crucial decisions about the proposed algorithm as well as work division were made. The key general approach was devised in these sessions and later detailed by specific team members.

Besides the shared team contributions (i.e. devising key algorithm ideas, cooperating on write-ups), some detailed contributions which were assigned as a result of team agreements can be given as:

- Makram Chahine - Impelementation of node embedding and all benchmarks; running all major test-cases; summarising results
- El Ghali Zerhouni - Key theoretical backing for the approach (i.e. preprocessing using first order and second order centrality); adjusting the node2vec algorithm to compute centralities and individual parameters p, q for each node; running small scale experiments
- Filip Ryzner - Literature reviews of current Deep Walk / Node2Vec modifications; Data collection and augmentation; Input processing for the algorithm; running small scale experiments

References

[1] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” 7 2016.

[2] W. L. Hamilton, R. Ying, and J. Leskovec, “Representation learning on graphs: Methods and applications,” pp. 1–23, 2017. [Online]. Available: http://arxiv.org/abs/1709.05584

[3] B. Perozzi, R. Al-Rfou, and S. Skiena, “DeepWalk: Online learning of social representations,” in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[4] S. Abu-El-Haija, R. Al-Rfou, B. Perozzi, and A. Alemi, “Watch your step: Learning node embeddings via graph attention,” in *Advances in Neural Information Processing Systems*, vol. 2018-Decem, 2018.

[5] Y. Dong, N. V. Chawla, and A. Swami, “Metapath2vec: Scalable representation learning for heterogeneous networks,” vol. Part F129685, 2017.

[6] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, “Struc2vec: Learning node representations from structural identity,” vol. Part F129685, 2017.

[7] “Link prediction with node2vec.” [Online]. Available: https://stellargraph.readthedocs.io/en/stable/demos/link-prediction/node2vec-link-prediction.html