WalkGAN: pairwise adversarial network alignment

Luis F. Guzman-Nateras
University of Oregon
lguzmann@uoregon.edu

Yang Zhou
Auburn University
yangzhou@auburn.edu

Dejing Dou
University of Oregon
dou@cs.uoregon.edu

***Abstract*—Network alignment (NA) consists on finding the optimal node correspondence between distinct networks (graphs). Previous works in this field have had various degrees of success. However, they rely on some strong assumptions of topological and/or attribute consistency among the aligned networks. Simultaneously, Generative Adversarial Networks (GANs), generative models that have achieved remarkable results on continuous data such as images and audio, have recently been successfully applied to tasks with discrete domains, such as text generation. This work presents our efforts into designing a GAN-based NA model to address the limitations of other approaches by exploiting (1) random walks' trait of being able to capture the local and global topological structure of networks and (2) GAN's ability to estimate the unknown distribution of the training data.**

***Index Terms*—network alignment, generative adversarial networks, node embeddings**

I. Introduction

Network Alignment (NA) consists in finding node correspondence across different networks. That is, for each node in a given network, find a node (or set of nodes) in a second network, that is (are) most likely to correspond to it. NA is often the first step in many data mining tasks [26] that analyze the complex structure and diverse relationships of several networks. Because of this, NA has attracted lots of attention from both the academic and industry environments [6]. Mapping pairs of similar nodes in different networks has many potential useful applications, such as user alignment in social networks. If the same (or similar) users can be identified in distinct social networks, for example, their profiles' characterization can be improved for more accurate classification or recommendations [23].

In this work, we focus on Pairwise Network Alignment (PNA)[1], which refers to finding the optimal alignment between a pair of networks. In contrast, Multiple Network Alignment (MNA) also aims to find an such an alignment, but between several (> 2) graphs[2]. PNA and MNA have each their own advantages and disadvantages. PNA is, usually, an easier problem to solve. However, in MNA there is more information available, which can lead to alignment results that would not be possible without multiple networks [6]. We restrict our work to PNA due to the nature of the proposed approach. Nonetheless, we do not discard it from being applicable to MNA (with some adjustments) in the future.

[1]When used in this paper, the term NA refers to PNA, unless stated otherwise.

[2]The concepts of network and graph are used interchangeably throughout this paper.

A. Motivation

Previous efforts in NA have had various degrees of success. Nonetheless, there are still areas for improvement. One example of this is the fact that some works [11, 21, 26] hold what is known as the *topology consistency* assumption. That is, nodes share a consistent connectivity structure across different networks. However, it is often observed that nodes corresponding to the same entities have very diverse neighbors and behaviors in different networks. Thus, they have quite different structural features. Say, for instance, two social networks with distinct semantics such as Facebook, which is mostly used for socializing with friends, and LinkedIn, which is used for job or professional purposes. This assumption can lead to sub-optimal or misleading alignments if it does not reflect the actual state of the networks.

Other related works [23, 26], rely heavily on attribute information to make precise alignments. However, this information may be unreliable, incomplete, or all-together missing which makes these methods not applicable in some instances. Many unsupervised NA algorithms rely heavily on highly discriminative attributes and node/edge types to ensure the effectiveness of the alignment, thus limiting their applicability. MEgo2Vec [25], for example, generates candidate pairs by considering only user pairs that share similar usernames. Yet, the same user does not always use similar usernames in different networks.

Finally, supervised NA methods use prior alignment knowledge in the form of *anchor nodes* (sometimes referred to as anchor links). These anchors are pairs of nodes (or sets in MNA) for which the alignment is already known (e.g. a user whose accounts are known in both Facebook and Twitter). The anchors are used for validation purposes and as alignment constraints to guide the alignment process. However, a large number of anchor node pairs are necessary to maintain the quality of the network alignment.

Concurrently to the research efforts in NA, Generative Adversarial Networks (GANs) were presented in [8]. GANs are frameworks to estimate generative models – models capable of generating data – by jointly training two distinct neural network models: a Generator and a Discriminator. They achieve this by proposing a minimax two-player game in which they pit the generator and discriminator against each other as adversaries. The purpose of the generator is to capture the distribution of the training data by producing data samples from a noise distribution. Meanwhile, the discriminator estimates the probability of a sample coming from the training data (also referred to as *real* samples) or from the samples created by the generator (*fake* samples). The generator is trained with the objective of maximizing the probability of fooling the discriminator. That is, making the samples created by the generator as similar to the real samples as possible. A unique equilibrium point for the proposed minimax game exists. In such equilibrium, the generator matches the training data distribution exactly, and the discriminator is unable to distinguish between the real and the fake samples.

GAN training is known to be complicated and unstable [1], making it difficult to reach the equilibrium solution. Nonetheless, GANs have been used to successfully estimate generative models capable of producing continuous data– such as images and audio–with some impressive results [1].

B. Contribution

Our objective is to design a semi-supervised NA method to align a pair of networks containing only topological information with few prior alignment knowledge.

Random walks have proved to be a powerful tool to capture both the local and global topological structure of networks. We argue that, it is highly possible for two nodes in different networks to belong to the same entity if multiple truncated random walks through them also go through many common anchor nodes. This implies these nodes share many common anchor nodes within their L–neighborhood, where L is the length of the truncated random walks. The reason to use truncated random walks with length L ($L > 1$), instead of using direct neighbors (i.e., $L = 1$), is that it is possible that there are no common anchor neighbors within 1–neighborhood when the amount of prior knowledge (anchor nodes) for training is limited, exacerbated by the assumption that the two networks have different topological features. On the other hand, by extending the radius of the neighborhood to L, there is a larger probability of finding common anchor neighbors within L–neighborhood since the two networks should still share some common structural features.

The main contribution of this work is to present a novel effort into using a GAN framework to tackle the problem of pairwise network alignment. Our intuition being that, by using random walks as training data, a successfully trained generator will capture the data distribution of a network. Thus, addressing the topology and attribute consistency assumptions that are widely embraced by other NA approaches, while still obtaining good alignment results even on networks with distinct semantics in which such assumptions do not hold.

II. Background and Related Work

We acknowledge as background, or related work, research efforts that fall into the following areas:

- Node embedding generation.
- Network alignment.
- GANs applied to discrete data.

A. Node embeddings

Recently, with the success attained using representation learning techniques such as *word2vec*[13] in domains like Natural Language Processing (NLP), network-embedding techniques have also been developed to learn latent network features [4, 9, 17, 22]. These techniques have been shown to be more effective at capturing the latent characteristics of a network [17, 23], and more efficient than their matrix-factorization based counterparts [9, 15].

DeepWalk [17] was probably the first graph representation-learning approach that gained popularity. DeepWalk learns latent representations of a graph's nodes by modeling a series of short, uniformly-sampled random walks. The use of random walks to extract information from a network has two main advantages: it's easy to parallelize, and it's easy to accommodate for small changes in the graph structure without having to recompute its representations. A node embedding captures its neighborhood similarity and community membership using the SkipGram [16] algorithm. SkipGram is a language modeling technique that aims to create word embeddings such that the vector features can predict nearby words. In a graph context, the embedding features should be able to predict the node's neighbors or its 'community'.

Node2Vec[9] is an extension upon the random walk approach to generating the network embeddings. Instead of using a uniform approach to generate the random walks, the authors specify a search bias α_{pq} which depends on two hyper-parameters (p and q) and affects the transition probabilities of the random walks. The p and q parameters guide the direction of the walk in the following way:

- p controls the likelihood of returning to the last visited node. Setting p to a high value makes it less likely to backtrack along the path, while setting a low p value would encourage the walk to remain "local".
- q allows the search to distinguish between "inward" and "outward" nodes. A higher q value makes the search biased towards nodes closer to the current node, while doing the opposite encourages outward exploration.

This approach to random walk generation proved to be more effective at tasks like multi-label classification and link prediction than the uniform generation approach (equivalent to $p = 1$, $q = 1$) used by DeepWalk.

There are other, non-random-walk-based approaches such as LINE[22], which optimizes two distinct loss functions: one approximating the first-order proximity between nodes, and another for the second-order proximity. First-order proximity is considered to be the local pairwise proximity between two nodes indicated by the weight of the edge between them. If there's no edge between two nodes then their local proximity is 0. Second-order proximity is the similarity between the neighborhood network structure of two nodes. The idea behind it is that if a pair of nodes have very similar neighborhoods then they must be similar to each other, even if there's no direct link between them.

B. Network alignment

Extensive research has been performed on NA. One common approach is to compute the structural alignment between networks by using matrix factorization (usually Eigenvalue decomposition) of the networks' adjacency matrix [2, 14, 21]. These approaches, however, usually involve taking the matrix inverse, which makes them costly and hard to scale up for large problems[15].

In FINAL[26] the authors propose a family of network alignment algorithms for attributed networks. Instead of relying only on the network's topology to find node correspondence, the authors argue that integrating attribute information from the nodes, which is usually present in many networks, can lead to better alignment results. This means they do not need to rely as much on the networks being topologically consistent, i.e., that the nodes display a consistent connectivity structure across networks.

REGAL[11] was the first work to propose node embeddings that generalize to multiple graphs. The approach to finding cross-network node similarity in REGAL is different from the singular network approaches [9, 17] because in these types of problems the nodes have no direct links between them and, thus, cannot be sampled by random walks. Instead, they define their similarity function in terms of structural identity and attribute-based identity. For structural identity, they build a vector using the degrees of the node's neighbors at varying distances. In the case of attribute-based identity, they create a vector representing the node's attribute values.

Another approach to network embedding generation and alignment is the IONE algorithm [15]. IONE encodes a node's follower-ship (incoming links) and followee-ship (outgoing links) into input and output context vectors, hence IONE (Input-Output Network Embedding). The idea being that the embedded space will preserve the proximity of nodes with similar sets of followers/followees. It also can use both known and suspected anchor users as hard and soft constraints to aid in contextual information transfer. Its objective function is built taking into account all these factors so that the embeddings are generated and the nodes are aligned simultaneously.

An MNA approach that also makes use of embedding representations is presented in CrossMNA[6]. Their approach consists on generating 3 different types of vectors. A *network* vector for each of the networks that reflects the similarity of global structure between networks. An *inter* vector for each node in a network that is shared across known anchor nodes and preserves the common features between such anchor nodes. And an *intra* vector for each node in the network that depicts the community features of a node within its selected network. They use the *inter* vectors to perform cross-network alignment.

In DeepLink[28], the authors propose to learn a soft bidirectional mapping between the embeddings of two networks using shallow neural networks and supervised learning. Then these soft mappings are jointly optimized in an unsupervised manner. Finally, the mappings are again improved via a supervised dual learning game in which each one tries to align the two latent spaces according to the rewards of mapping the anchor nodes.

The approach presented in SNNA[5] projects the distribution from a source network into the distribution of a target network using an adversarial learning framework. In their work, the authors rely on attribute information and feature extraction besides pure strutural information when creating the latent space. They use a modified WGAN loss and test three versions of their approach: unidirectional, bidirectional, and orthogonal projection model with this latter one obtaining the best performance.

C. GANs for discrete data

Even though GANs were originally envisioned to work with continuous data such as images, there have been efforts into applying GAN-like architectures to deal with discrete data, particularly in Language Modeling (LM). Language models estimate the probability of a sequence of words and can be used to generate text sequences by sampling from these estimated distributions.

In TextGAN[27], the authors present an approach that uses an LSTM-based RNN generator with a CNN discriminator to produce text sequences at the word level. They use the standard GAN loss function but propose an approximate discretization to select the next character in the sequence at inference time.

A similar effort is presented in [19]. The authors, however, explore both CNN and RNN architectures for the discriminator and the generator. Additionally, they use the WGAN loss function along with curriculum learning to train their models.

The authors of [18] use the WGAN loss function, along with an RNN-based generator and discriminator. They work at the character level to generate the text sequences but do not perform any sampling at training time. Instead, they feed the discriminator with 'soft' distribution vectors as the 'fake' sequences, and one-hot encoded vectors for the real ones.

Another, more recent, effort into adversarial text generation is the MaskGAN model presented in [7]. In their work, the authors use sequence-to-sequence models (encoder-decoder pairs) for both the generator and discriminator. They use the WGAN loss function and use reinforcement learning to estimate the gradient updates for the generator. Another key component of their work, is that they "mask" some of the words in the sentence and their objective is to train their models to predict these masked elements based on their context.

Finally, NetGAN[3], though not a language model, is a different example of a GAN architecture applied to discrete data. The authors tackle the task of reconstructing a graph by training a discrete GAN model to learn to generate sequences of nodes using random-walks, taken from the original graph, as the training data. In their work, the authors use an RNN-based generator and discriminator. They also use Gumbel-Softmax reparametrization to deal with the flow of gradients during backpropagation.

We consider NetGAN to be the work that is most closely related to ours. We adopt some of the fundamental ideas presented in both MaskGAN[7] and NetGAN[3] and combine them into our proposal which is described in section III.

III. WalkGAN: an adversarial network alignment approach

A. Problem definition

A pairwise network alignment problem has two networks denoted as $G^k = (V^k, E^k)(1 \leq k \leq 2)$, where $V^k = v_1^k, \ldots, v_{N^k}^k$ is a set of nodes with cardinality $|V^k| = N^k$, and $E^k = \{(v_i^k, v_j^k) : 1 \leq i, j \leq N^k, i \neq j\}$ is a set of edges. A node $v_i^k \in V^k (1 \leq i \leq N^k)$ represents an entity in G^k (*e.g.,* a person in the Facebook graph). An edge $(v_i^k, v_j^k) \in E^k$ is associated with two nodes v_i^k and v_j^k, and denotes a relationship between two entities (*e.g.,* two persons that are friends on Facebook would have an edge that links their corresponding nodes). The objective of the pairwise alignment problem is to match nodes $v_i^1 \in G^1$ and $v_j^2 \in G^2$ that belong to the same entity (*e.g.,* given a known person on Facebook, find the same person on Twitter).

In addition, in network alignment problems, it is usual to have a set of **anchor-node** pairs $Anc = \{(v_i^1, v_j^2) | v_i^1 \leftrightarrow v_j^2\}$, where v_i^1 and v_j^2 are the anchor nodes in G^1 and G^2 respectively, and $v_i^1 \leftrightarrow v_j^2$ indicates that v_i^1 and v_j^2 belong to the same entity. That is, the set of anchor nodes Anc consists on pairs of nodes for which the corresponding alignment is already known. These pairs of already aligned nodes can then be used as ground truth for validation purposes, or to guide the alignment process.

When performing the alignment, nodes are represented as vectors or node embeddings. Using a vector representation allows nodes to be compared using distance metrics, and to be used as inputs for the neural network architecture. A simple way to define the vector representation of a given node v_i^k is to use the corresponding i^{th} row of the adjacency matrix $\mathbf{A}^k$ of G^k. $\mathbf{A}^k$ is an $N^k \times N^k$ matrix where the value at the i^{th} row and j^{th} column, denoted as $\mathbf{A}_{ij}^k$, specifies the value of the edge between nodes v_i^k and v_j^k. The symbol $\mathbf{A}_i^k$ can then be used to denote the i^{th} row vector of $\mathbf{A}^k$, which is used to specify the vector representation of the node v_i^k. For ease of presentation, we use v_i^k to denote the node itself and its vector representation $\mathbf{A}_i^k$. This simple approach, however, leads to representations that are very sparse due to their high dimensionality ($A_i^k \in \mathbb{R}^{N^k}$). Instead, we utilize existing, representative node embedding approaches, such as DeepWalk [17] and Node2Vec [9], to learn the pre-trained individual embeddings of the nodes in G^1 and G^2, respectively. These techniques map the vector representation $\mathbf{A}_i^k$ of each node v_i^k to a low-dimensional embedding u_i^k with the minimum reconstruction error between the output and the input, *i.e.,* $v_i^k : \mathbb{R}^{N^k} \mapsto u_i^k : \mathbb{R}^D$ and $D << N^k$. The alignment of the networks is then performed in terms of these low-dimensional embedding features u_i^1 and u_j^2.

B. Random Walk Generation

Random walks have been shown to be able to capture the structural features of a graph [9, 17]. A Random-Walk Generator (RWG) receives a network G^k as an input and produces a finite random walk through the network as the output. For our experiments, we tried with two distinct random-walk generation strategies. Both aim to make multiple random walks capture as many topological features of the anchor nodes $v_i^1 \in Anc$ as possible.

Both strategies use G^1 as the input network and uniformly sample a random anchor node $v_i^1 \in Anc$ as the starting point of a walk W_i^1. We then repeatedly generate R random walks of length L beginning from v_i^1. These walks are denoted as $W_{i1}^1, \ldots, W_{iR}^1$. We use the symbol W_{irl}^1 to represent the $l^{th}(1 \leq l \leq L)$ node in the $r^{th}(1 \leq r \leq R)$ walk starting from v_i^1. Notice that $W_{ir1}^1 = v_i^1$.

Our random-walk generation strategy, here-on denoted as RWG^1, is summarized in the following algorithm:

Algorithm 1 Random-walk generation strategy 1

```
1: procedure RWG$^1$($W_{irl}^1$, $Anc$)
2:     $W_{ir}^1 \leftarrow [W_{irl}^1]$
3:     $l \leftarrow 0$
4:     while $|W_{ir}^1| < L$ do
5:         $v_l^1 \leftarrow W_{irl}^1$
6:         if Neighbors($v_l^1$) $\cap Anc \cap \overline{W_{lr}^1} \neq \emptyset$ then
7:             $v_{l+1}^1 \leftarrow$ sample($N(v_l^1) \cap Anc \cap \overline{W_{lr}^1}$)
8:         else
9:             if Neighbors($v_l^1$) $\cap \overline{W_{lr}^1} \neq \emptyset$ then
10:                $v_{l+1}^1 \leftarrow$ sample($N(v_l^1) \cap \overline{W_{lr}^1}$)
11:            else
12:                Discard $W_{ir}^1$
13:            end if
14:        end if
15:        $W_{ir}^1$.append($v_{l+1}^1$)
16:        $l = l + 1$
17:    end while
18:    return $W_{ir}^1$
19: end procedure
```

where Neighbors(v) is a function that returns the set of nodes that share an edge with node v, and $\overline{W_{lr}^1}$ is the complement of the set of nodes already in W_{lr}^1. The RWG algorithm prioritizes to randomly sample an unvisited anchor neighbor of the last visited node v_l^i, if there are any available. Otherwise, it uniformly samples from the unvisited non-anchor neighbors of v_l^i. These steps are repeated until the desired length L of the walk is reached, or all neighbors of v_l^i have been visited during the generation of W_{ir}^1 and the current, partial walk is discarded. This process is replicated for every anchor node in A until the desired number R of walks per anchor node is achieved.

Our RWG^1 approach encourages each random walk to go through as many anchor nodes as possible, while also visiting distinct nodes and avoiding creating loops. This makes the radius of the subgraph (*i.e.,* path of W^1_{ir}) as large as possible.

Similarly, our second random-walk generation strategy (RWG^2) is summarized in the following algorithm:

Algorithm 2 Random-walk generation strategy 2

```
1:  procedure RWG$^1$($W^1_{irl}$, $Anc$)
2:      $W^1_{ir} \leftarrow [W^1_{irl}]$
3:      $l \leftarrow 0$
4:      while $|W^1_{ir}| < L$ do
5:          $v^1_l \leftarrow W^1_{irl}$
6:          if $v^1_l \in Anc$ then
7:              if Neighbors($v^1_l$) $\cap \overline{W^1_{lr}} \neq \emptyset$ then
8:                  $v^1_{l+1} \leftarrow$ sample($N(v^1_l) \cap \overline{W^1_{lr}}$)
9:              else
10:                 Discard $W^1_{ir}$
11:             end if
12:         else
13:             if Neighbors($v^1_l$) $\cap Anc \cap \overline{W^1_{lr}} \neq \emptyset$ then
14:                 $v^1_{l+1} \leftarrow$ sample($N(v^1_l) \cap Anc \cap \overline{W^1_{lr}}$)
15:             else
16:                 Discard $W^1_{ir}$
17:             end if
18:         end if
19:         $W^1_{ir}$.append($v^1_{l+1}$)
20:         $l = l + 1$
21:     end while
22:     return $W^1_{ir}$
23: end procedure
```

The main change in our RWG^2 strategy consists in not allowing two non-anchor nodes to occur consecutively in a walk. If the current node in the walk is not an anchor, the next node must be sampled from the anchors. If there does not exist a suitable anchor node, the walk is discarded. The intuition behind RWG^2 is that, by only having single non-anchor nodes between pairs of anchors, the alignment process can be simplified. There are, however, some drawbacks to RWG^2 when compared with RWG^1 that arise from the additional restrictions:

- Fewer random walks can be generated starting from each available anchor node which, in turn, means that there is less data available for training.
- The number of encountered non-anchor nodes is greatly reduced. Consequently, there are fewer candidates for alignment which limits the overall impact of our approach. This, nonetheless, can also be a considered as a favorable characteristic since the non-anchors, that do appear, are encountered more frequently and can be aligned with more certainty.

The set W^1, with cardinality $|W^1| = R * |Anc|$, of generated random walks, using either RWG^1 or RWG^2, serves as the data set for our experiments.

As each walk $W^1_{ir} = (W^1_{ir1}, \dots, W^1_{irL})$ in W^1 is generated, a **binary mask** $M_{ir} = (M_{ir1}, \dots, M_{irL})$ of the same length is concurrently produced. Each element M_{irl} in a mask has a value of 1, if W^1_{irl} is an anchor node, or a value of 0, otherwise.

Using a walk W^1_{ir} and its mask M_{ir}, we then create a semi-translated walk: given that the correct alignment for anchor nodes is already known, every node in W^1_{irl} with mask $M_{irl} = 1$ is replaced by its corresponding node in G^2. All other nodes (those with mask $M_{irl} = 0$) are replaced by an empty identifier ϕ representing non-anchor nodes in the walk. This procedure yields a **masked walk** $W^2_{ir} = (W^2_{ir1}, \dots, W^2_{irL})$ in G^2 that gets its name due to the fact that it's composed by translated anchor nodes and masked non-anchor nodes. Figure 1 illustrates the aforementioned procedure.

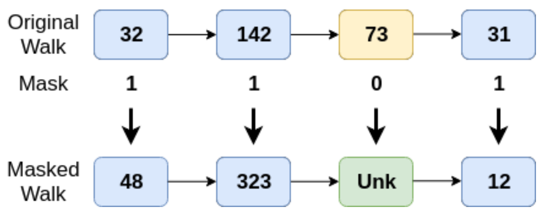


Fig. 1: Masking procedure: anchor nodes are colored blue, non-anchors in yellow, and masked nodes in green.

C. GAN Model

The objective of our WalkGAN architecture is to match pairs of nodes in the two networks in terms of both their local and global structures. In this section we discuss the design details of our model.

The generator in our GAN model is composed by two distinct sub-models: the filler $\mathbf{F}$ and the translator $\mathbf{T}$. The filler receives a masked walk W^2_{ir} as input, and fills in the masked nodes, identified by ϕ, with suitable nodes in G^2. Thus, its output is a **filled walk** $\hat{W}^2_{ir} = \mathbf{F}(W^2_{ir})$. These filled walks $\hat{W}^2$ serve as inputs to the translator model $\mathbf{T}$ which works as an alignmenter and outputs **translated walks** $\hat{W}^1_{ir} = \mathbf{T}(\hat{W}^2_{ir})$ in G^1. The generator $\mathbf{G}$, as a whole, can then be defined as $\mathbf{G}(W^2_{ir}) = \mathbf{T}(\mathbf{F}(W^2_{ir})) \rightarrow \hat{W}^1_{ir}$ (Figure 2).

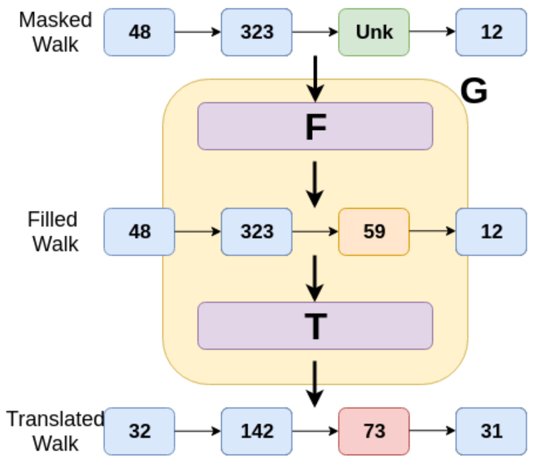


Fig. 2: Two-step Generator architecture.

The purpose of the discriminator $\mathbf{D}$ is to output the probability that a random walk belongs to either the set of real random walks W^1 sampled from G^1, or the fake random walks $\hat{W}^1$ created by the generator after filling and translating the masked walks W^2.

The loss function for our GAN model is defined as the following minimax optimization problem:

$$\min_{\mathbf{F},\mathbf{T}} \max_{\mathbf{D}} \mathcal{U}(\mathbf{D}, \mathbf{T}, \mathbf{F}) = \mathbb{E}_{i \sim p(i)} \mathbb{E}_{r \sim p(r)} [\log \mathbf{D}(W^1_{ir}) - \log \mathbf{D}(\mathbf{T}(\mathbf{F}(W^2_{ir})))] \tag{1}$$

The filler $\mathbf{F}$ computes and decomposes the distribution over the whole random walk into the distribution of each non-anchor node in the walk given the context of the mask M_{ir} (2).

$$\begin{aligned} \mathbf{F}(W^2_{ir}) &= P(\hat{W}^2_{ir} | W^2_{ir}, M_{ir}) \\ &= P(\hat{W}^2_{ir1}, \ldots, \hat{W}^2_{irL} | W^2_{ir}, M_{ir}) \\ &= \prod_{l=1}^{L} P(\hat{W}^2_{irl} | \hat{W}^2_{ir1}, \ldots, \hat{W}^2_{ir(l-1)}, M_{ir}) \end{aligned} \tag{2}$$

Where each distribution $P(\hat{W}^2_{irl} | \hat{W}^2_{ir1}, \ldots, \hat{W}^2_{ir(l-1)}, M_{ir})$ is defined as a softmax over all nodes in G^2.

Let $\mathbf{F}' = P(\hat{W}^2_{irl} | \hat{W}^2_{ir1}, \ldots, \hat{W}^2_{ir(l-1)}, M_{ir})$, which denotes the filler at the node level, we can rewrite (2) as:

$$\mathbf{F}(W^2_{ir}) = \prod_{l=1}^{L} \mathbf{F}'(W^2_{irl}) \tag{3}$$

The discriminator $\mathbf{D}$ computes and decomposes the probability of a sample random walk, W^1_{ir} or $\hat{W}^1_{ir}$, being a real walk into the multiplication of the probability of each non-anchor node being real in the context of the mask M_{ir} (4). $\mathbf{D}$ has the same framework as $\mathbf{F}$. The difference is it outputs a probability of a sample random walk being real, rather than a distribution over the node set V^2 and a filled node with the maximum score.

$$\begin{aligned} \mathbf{D}(W^1_{ir}) &= \prod_{l=1}^{L} P(W^1_{irl} = \mathit{real} | W^1_{ir}, M_{ir}) \\ \mathbf{D}(\hat{W}^1_{ir}) &= \prod_{l=1}^{L} P(\mathbf{T}'(\mathbf{F}'(W^2_{irl})) = \mathit{real} | W^2_{ir}, M_{ir}) \end{aligned} \tag{4}$$

Where $\mathbf{T}'$ is the translator at node level, *i.e.,* translates a node in a filled walk $\hat{W}^2_{ir}$ in G^2 into a node in G^1.

Similarly, let $\mathbf{D}'(W_{irl})$ denote the discriminator at node level, we can then reformulate (4) as:

$$\begin{aligned} \mathbf{D}(W^1_{ir}) &= \prod_{l=1}^{L} \mathbf{D}'(W^1_{irl}) \\ \mathbf{D}(\hat{W}^1_{ir}) &= \prod_{l=1}^{L} \mathbf{D}'(\hat{W}^1_{irl}) \end{aligned} \tag{5}$$

Therefore, we reorganize the original GAN model at the walk level into another GAN model at the node level:

$$\min_{\mathbf{F}',\mathbf{T}'} \max_{\mathbf{D}'} \mathcal{U}(\mathbf{D}', \mathbf{T}', \mathbf{F}') = \mathbb{E}_{i \sim p(i)} \mathbb{E}_{r \sim p(r)} [\log \mathbf{D}'(W^1_{ir}) - \log \mathbf{D}'(\mathbf{T}'(\mathbf{F}'(W^2_{ir})))] \tag{6}$$

Figure 3 presents a high-level view of the complete WalkGAN architecture: the generator $\mathbf{G}$ receives the masked walks as input and outputs translated walks. These translated walks are then fed to the discriminator $\mathbf{D}$ along with the real walks for it to produce a prediction. Next, the prediction errors are used to update the parameters of both the discriminator and the generator. This procedure is iteratively repeated until the model converges or a pre-determined maximum number of iterations is reached.

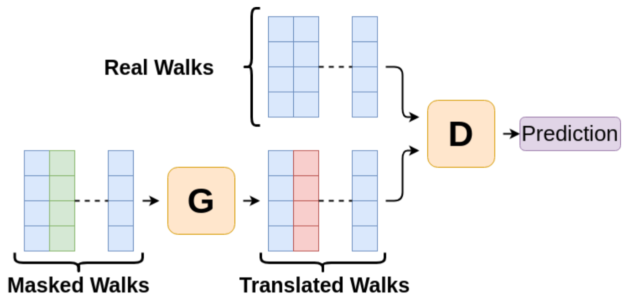


Fig. 3: WalkGAN architecture.

D. *Model framework*

As mentioned in the previous section, the frameworks for $\mathbf{F}$ and $\mathbf{D}$ are very similar. In both cases, we use an RNN-based encoder-decoder model: the encoder receives a masked random walk as input, and outputs the last hidden state of the RNN model as the latent representation for the input walk. The decoder uses the latent representation obtained by the encoder to initialize its own hidden state and then produces its desired output. The output of the decoder is where the key difference between $\mathbf{F}$ and $\mathbf{D}$ relies: the former outputs a probability distribution over the node set V^2, while the latter outputs the probability of a given walk being real, i.e. that it comes from W^1.

We use Long Short-Term Memory (LSTM) cells for our RNNs. All vectors in the LSTM model have the same dimension D as the node embedding vectors. The LSTM cells can be replaced with a different RNN model such as Gated Recurrent Units (GRU) for enhancing the efficiency since they usually display similar performance.

Without loss of generality, here we describe in greater detail the LSTM architecture of the filler $\mathbf{F}$. First, the encoder receives a masked random walk $W^2_{ir} = (W^2_{ir1}, \ldots, W^2_{irL})$ along with its corresponding binary mask $M_{ir} = (M_{ir1}, \ldots, M_{irL})$ as input, and outputs its last hidden state h_L as the latent representation of W^2_{ir}. The decoder receives this representation vector and uses it to initialize its own initial hidden state h_0. Next, it computes the conditional probability

$P(\hat{W}^2_{ir1}, \ldots, \hat{W}^2_{irL}|W^2_{ir}, M_{ir})$ node by node using the Softmax function. The Softmax function (7) operates on a context vector h_l (*i.e.,* hidden state) and a node embedding vector u^k_i to compute the distribution of each non-anchor node in the random walk over all nodes in G^2.

$$P(\hat{W}^2_{ir1}, \ldots, \hat{W}^2_{irL}|W^2_{ir}, M_{ir}) = \frac{\exp(h_l^T \cdot u_j^2)}{\sum_{k=1}^{N^k} \exp(h_l^T \cdot u_k^2)} \quad (7)$$

During training, whenever a non-masked node, *i.e.,* anchor node, is encountered during the filling process, it is left untouched by the decoder: its embedding is appended to the resulting filled walk $\hat{W}^2_{ir}$ and used as the input for the next step in the process, independently of whatever the decoder outputs. A masked node, however, is filled with the embedding u^2_j of the node $v^2_j \in V^2$ that has the maximum Softmax value and becomes an unmasked node $\hat{W}^2_{irl}$. By repeating this process for every masked node in the walk, the filler outputs the filled walk $\hat{W}^2_{ir}$.

The translator at the node level $\mathbf{T}'$ is implemented by adding an additional linear layer to translate each filled non-anchor node $\hat{W}^2_{irl}$ in a filled walk $\hat{W}^2_{ir}$ in G^2 into a corresponding non-anchor node $\hat{W}^1_{irl}$ in the translated walk $\hat{W}^1_{ir}$ in G^1.

$$\hat{W}^1_{irl} = \mathbf{T}'(\hat{W}^2_{irl}) = W_T \cdot \hat{W}^2_{irl} \quad (8)$$

where $W_T \in \mathbb{R}^{D \times D}$ is a translation matrix to be learned.

A fill and translation loss between the original masked non-anchor nodes and the corresponding filled and translated non-anchor nodes is used to improve the quality of the filling and translation functions.

$$\mathcal{L}_{trans} = \quad (9)$$
$$\min_{\mathbf{T}',\mathbf{F}'} \sum_{v^1_i \in Anc} \sum_{r=1}^{R} \sum_{l=1}^{L} ||W^1_{irl} - \mathbf{T}'(\mathbf{F}'(W^2_{irl}))||_2, \text{if } M_{irl} = 0$$

E. Dealing with sampling

Unlike the generators in traditional GAN models which generate fake images from a continuous noise space, the process of unmasking a masked node W^2_{irl}, by selecting a suitable real node, performed by the filler $\mathbf{F}$ is discrete. At each time step, the filler samples from the estimated probability distribution over all the nodes in V^2 in order to obtain the next node in the sequence. However, sampling from a categorical distribution is a non-differentiable operation which prevents the use of the backpropagation algorithm to propagate the gradients through the model and train it using the standard Stochastic Gradient Descent (SDG) method [3].

Different approaches have been used to deal with the issue of having a generator produce discrete data. The authors of MaskGAN[7] use reinforcement learning to approximate the generator's gradient updates. In NetGAN [3] they make use of the Gumbel-Softmax reparametrization trick [12]. Other efforts such as [10, 18] propose not sampling at all during training and only doing it during inference time for evaluation purposes. In this work, we tried with two approaches: initially, a policy gradient[20] based reinforcement learning approach, and eventually, an approach using Gumbel-Softmax reparametrization.

For our initial reinforcement-learning approach, we estimate the parameters updates of $\mathbf{F}'$ by performing gradient ascent using an algorithm from the REINFORCE family:

$$\nabla_\theta \mathbf{F}'[R_l] = R_l \nabla_\theta \log \mathbf{F}'_\theta(W^2_{irl}) \quad (10)$$

Similar to the approach in [7], a node generated at step l not only defines the reward R_l, but also influences all future rewards:

$$\nabla_\theta \mathbf{F}'[R_l] = \sum_{l=1}^{L} (R_l) \nabla_\theta \log \mathbf{F}'_\theta(W^2_{irl}) \quad (11)$$
$$= \sum_{l=1}^{L} (\sum_{s=l}^{L} \gamma^s r_s) \nabla_\theta \log \mathbf{F}'_\theta(W^2_{irl}) \quad (12)$$

Where γ is non-zero discount factor that penalizes greedily selecting a node that earns immediate high-rewards alone. We define the rewards r_s as follows:

$$r_s = \log \frac{1}{1 - \mathbf{D}'(\mathbf{T}'(\hat{W}^2_{irs}))} \quad (13)$$

As mentioned previously, for our second approach we use the Gumbel-Softmax transformation:

$$v^*_l = \sigma(\frac{\mathbf{p}_l + \mathbf{g}}{\tau}) = \sigma(\frac{\mathbf{F}'(W^2_{irl}) + \mathbf{g}}{\tau}) \quad (14)$$

where σ is the softmax funtion, $\mathbf{g}$ are samples from a Gumbel distribution with zero mean and scale of 1, $\mathbf{p}_l$ is the probability distribution estimated by the RNN generator at time l, and τ is a temperature parameter that controls the trade-off between better flow of gradients (large τ) and more exact calculations (small τ). Then, the next node in the sequence is obtained by taking the argmax, and encoding it as a one-hot vector.

$$v_l = one\_hot(argmax(v^*_l)) \quad (15)$$

This one-hot encoded v_l is used as the input for the next time step. However, during the backward pass, the gradients flow through the differentiable v^*_l [3].

F. Wasserstein GAN

The standard GAN loss function seeks to improve the discriminator's ability to distinguish between real and fake data while, at the same time, improving the generator's ability to trick the discriminator. The aforementioned, creates a zero-sum game that forces both to improve their functionalities. Mathematically, this is expressed as a minimax game in which the following loss function is optimized:

$$\min_G \max_D L(D, G) = \quad (16)$$
$$\mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

If the generator is trained to its optimal point, then $p_g = p_r$ and $D^*(x) = 0.5$, *i.e.,* the discriminator is not able to tell the real data apart from the fake one. In practice, however, achieving this point of equilibrium is difficult since each model updates its cost independently, and updating both models concurrently does not guarantee convergence. Because of this, GAN training is known to be delicate and unstable [1].

Two of the most common problems that arise during the training phase are vanishing gradients[3] and mode collapse. Vanishing gradients occur when the discriminator quickly learns to distinguish between real and fake data which, in turn, causes it to provide unreliable gradient information to the generator. Mode collapse, on the other hand, happens when the generator always produces the same outputs, which usually means that the generator is being able to fool the discriminator by producing 'garbage'. Thus, in a sense, vanishing gradients and mode collapse can be seen as different sides of the same coin: the former takes place when the discriminator is too good at its job, while the latter happens if it's not good enough.

A proposed approach to deal with the training issues mentioned above is to use a Wasserstein GAN (WGAN)[1], which has been known to lead to more stable training overall and prevents mode collapse [3]. WGAN uses the Wasserstein distance, also called Earth Mover's distance, which is continuous and differentiable, to calculate the loss function for the generator and discriminator:

$$L_D = \mathbb{E}[D(x)] - \mathbb{E}[D(G(z))] \tag{17}$$

$$L_G = \mathbb{E}[D(G(z))] \tag{18}$$

The job of the discriminator in a WGAN is different: it is no longer tasked with telling fake samples apart from the real ones. Instead, it learns a continuous function to help compute the Wasserstein distance. For this reason, the authors in [1] call it the *critic* instead. As the loss decreases during training, the Wasserstein distance gets smaller which, in turn, means that the outputs from the generator grow closer to the real data distribution. To enforce Lipschitz continuity, the authors propose clipping the weights to a small window ([-0.01, 0.01]) resulting in a compact parameter space, however, they also state that weight clipping is a terrible way to enforce a Lipschitz constraint. Thus, a variation of the standard WGAN was introduced in [10] where, instead of clipping the gradients, a **gradient penalty** is computed to enforce the Lipschitz constraint:

$$\begin{aligned} \tilde{x} &= G(z) \\ \hat{x} &= \epsilon x + (1-\epsilon)\tilde{x} \\ L_D &= \mathbb{E}[D(x)] - \mathbb{E}[D(\tilde{x})] + \lambda(||\Delta_{\hat{x}} D(\hat{x})||_2 - 1)^2 \end{aligned} \tag{19}$$

where ϵ is a random number sampled from a uniform distribution $U[0,1]$, and λ is called the penalty coefficient ($\lambda = 10$ by default).

IV. Experiments and current results

This section includes a subset of the experiments we have performed thus far and the corresponding intermediate results. It is meant to showcase the progression of our work and provide a revision of the distinct steps we have taken in an attempt to overcome the difficulties encountered when implementing our model.

A. Data sets

We have three real wold data sets available to use for our experiments. Their characteristics are shown in the following table:

TABLE I: Data set characteristics

	# nodes			
Data set	Graph 1	Graph 2	Graph 3	**# anchors**
Autonomous systems	10,900	11,113	11,461	10,580
DBLP	28,478	26,455	-	25,926
Social networks	5,742	5,053	4,829	588

The autonomous systems dataset[4] represents information from router networks in the state of Oregon. The DBLP data set composed of co-authorship networks taken from the computer science bibliography website[5]. The social networks dataset is composed by graphs taken from three distinct social networks: Flicker, Livejournal, and Myspace. The fact that we have an almost complete prior alignment knowledge for the autonomous systems (92% known anchor pairs) and the DBLP (91% know anchor pairs) data sets allows us to quantitatively measure the performance of our approach.

We chose to develop our model using the autonomous system data set given that its networks are medium-sized ($\sim$ 10K nodes) and not overly sparse. Hence, all of the experiments presented in this section were performed using such data set as training and validation data.

It is our intention to test our model's performance on the other two data sets once we have tuned our parameters and obtained a sufficiently acceptable performance.

B. Model evaluation

The alignment of the networks comes from successfully training the generator $\mathbf{G}$ to (1) correctly impute the masked nodes and (2) correctly translate the embeddings of the imputed nodes in G^2 to its corresponding node's embeddings in G^1. Thus, we can actually evaluate our model's performance using either the filler $\mathbf{F}'$ or the translator $\mathbf{T}'$.

However, evaluating $\mathbf{F}'$ is more complicated due to the fact that it's either *hit-or-miss*, i.e. either $\mathbf{F}'$ imputes the correct node or not. Additionally, $\mathbf{F}'$ will impute different G^2 nodes for the same G^1 node in distinct walks depending on the context.

Furthermore, NA methods are usually evaluated using *top-k* precision [6, 11, 15] – a node is considered as correctly aligned if its ground-truth alignment is found in the top *k* predictions – using some sort of alignment score.

For these reasons, we choose to evaluate using the translator $\mathbf{T}'$ instead. By using $\mathbf{T}'$ we can simply compare its output embeddings with the expected embeddings using some distance metric, e.g. Cosine distance.

[3]Even though they share a name, it is important to note that this is a different phenomenon than the one that occurs in RNNs that deal with long sequences.

[4]https://snap.stanford.edu/data/Oregon-2.html

[5]https://dblp.uni-trier.de/

This, however, implicitly sets an upper bound on the alignment results we can obtain because of there's a limit on how good of a mapping between the two embedding sets can be learned by the $\mathbf{T}'$ module [28]. To test these assumption we trained a $\mathbf{T}'$ using standard SGD and Cosine Embedding loss with varying amounts of anchor nodes which is analogous to having $\mathbf{F}'$ correctly impute such nodes. Then, we compute the alignment accuracy for 5 different values of $k = \{5, 10, 30, 50, 100\}$ using a KD-Tree. The results are presented in Figure 4.

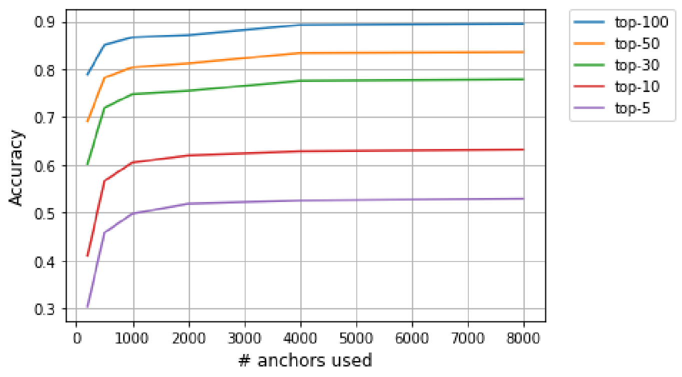


Fig. 4: Single-layer $\mathbf{T}'$ *top-k* alignment accuracy

There are several insights obtained from these results:

- When considering the top-100 predictions, the alignment accuracy is high (0.7891), even when using very few (200) anchors to train. This result implies that the networks are pretty similar, given that really good alignment results can be obtained by mapping only a few pairs of anchor embeddings.
- Accuracy never reaches 90% even when using lots of training anchors ($8K$) and using top-100 predictions.
- Top-5 accuracy plateaus at around 55%, even when using $+5K$ anchors to train, which seems counter-productive given that at $+5K$ we already have around 50% top-1 accuracy.
- All of the curves seem to plateau when using around $4K$ nodes ($\sim 40\%$ of the graph) without further significant gains.

These observations confirm our assumptions about the upper bound performance of our model. On the other hand, they also imply that we can achieve results close to that upper bound by obtaining around 40% anchor pairs.

By replacing the single-layer $\mathbf{T}'$ with a slightly deeper model with an extra hidden layer and a non-linearity, the alignment accuracy results can be improved (Figure 5). However, all of the previous observations are still valid. Further 'deepening' $\mathbf{T}'$ does not seem to improve the results (Appendix A).

C. Module pre-training

Before discussing training of the GAN framework as a whole, we first we talk about model pre-training, as it is common in GAN-based works [7, 24]. We can individually

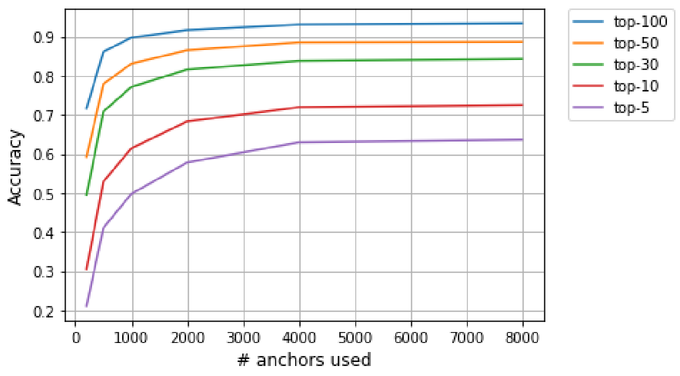


Fig. 5: Multi-layer $\mathbf{T}'$ *top-k* alignment accuracy

pre-train the three modules that make up our model: filler $\mathbf{F}'$, translator $\mathbf{T}'$, and discriminator $\mathbf{D}$.

The role of $\mathbf{F}$ is to impute the masked walks W^2 with suitable nodes in G^2, thus outputting the filled walks $\hat{W}^2$. Thus, a good starting point is to have $\mathbf{F}$ generate valid random walks in G^2. This is analogous to how a language model is trained to generate text sequences. The training corpus is obtained by sampling a considerable amount $(+500K)$ of truncated random walks through G^2. These sampled walks have the same length as the length of the masked random walks W^2 that $\mathbf{F}$ imputes when performing the alignment. Using the generated set of walks as training/validation data, we then train $\mathbf{F}$ via the standard Maximum Likelihood Estimation (MLE) approach until a satisfactory validation error is achieved. We found that it is critical for a better performance that we mask some nodes in the input walks to mimic the masked nodes the encoder would encounter during GAN training. This way the encoder learns to include information about the masked nodes in its output hidden state.

The translator $\mathbf{T}$, as discussed previously, aims to translate the embedding vector of a given node v_i^2 to the embedding vector of its corresponding pair v_j^1. Given that we have a set of anchor pairs (Anc) available, we can pre-train $\mathbf{T}$ to minimize the error when translating between the pairs $(v_j^1, v_i^2) \in Anc$. We utilize a subset of the known anchor-node set Anc for training – around 90%– while the rest is left for validation purposes. We use the Cosine Embedding distance to measure the similarity of the embeddings produced by $\mathbf{T}$ and the expected embedding vectors. We found that the cosine distance achieved a better performance than other similarity metrics like Mean Squared Error (MSE).

Finally, $\mathbf{D}$'s pre-training consists in getting it to distinguish between actual random walks in G^1 and random noise sampled from a Gaussian distribution. For this task, we use standard Binary Cross Entropy (BCE) loss. We provide $\mathbf{D}$ with embedded random walks from the W^1 set as positive examples. For negative examples, we create fake walks out of noise vectors with the same dimensionality and length as the embedded W^1 walks.

D. Parameters and model configuration

In this section we provide a rundown of the results we have obtained and an in-depth analysis of the GAN training progress. For the experiments shown in this section we use the following set up:

- Autonomous Systems data set using 'o1' as G^1 and 'o2' as G^2.
- 128-dimension Node2Vec embeddings with $p = 1$ and $q = 0.5$.
- Random-walk length = 9.
- 500 anchor-node pairs ($\sim 5\%$)
- Single-layer, bidirectional LSTMs with 128-dimension hidden states for the encoders and decoders.
- Multi-layer $\mathbf{T}'$ with 128 input/output dimentions, and a hidden layer with 512 neurons and ReLU activation.

We found that this model configuration yields the best performance. Additional results with different model configurations and hyper-parameters can be found in Appendix A.

E. Alignment before training

Once the filler $\mathbf{F}$ has been pre-trained, we tested its performance at imputing the masked walks W^2 before starting the GAN training. Of course, $\mathbf{F}$ chose different G^2 nodes to fill-in for the same G^1 node in different walks. So, we devised the following strategy to choose the best alignment candidate:

- Choose the G^2 node with the most appearances.
- In case of a tie, choose the candidate with the highest anchor-node average.

Table II shows the alignment results obtained by following this selection strategy. Accuracy is fairly low for both random walk generation modes. RWG^2 is almost twice as accurate, however, due to the restricted way in which walks are generated. One interesting observation is that $\mathbf{F}$ is considerably less certain when using RWG^2 given that the amount of G^2 nodes used is almost double the number of encountered G^1 nodes.

TABLE II: Alignment results before training

Mode	Walks	G^1 nodes	G^2 nodes	Correct	Acc
RWG^1	48096	3483	3926	405	11.6%
RWG^2	39782	973	1711	221	22.7%

Upon further inspection, we discovered that many of the encountered nodes appear only a handful of times in the walks. Given that our selection strategy is based on number of appearances, the alignment for these less common nodes is not very trust-worthy. Table III shows the change in alignment results by only considering nodes that appear > 100 times:

TABLE III: Alignment for non-anchor nodes appearing > 100 times

Mode	G^1 nodes	Correct	Acc	Recall
RWG^1	341	219	64.2%	54%
RWG^2	267	149	55.8%	67.4%

When only considering nodes that appear frequently (> 100 times) in the walks, the alignment accuracy increased significantly: 53% for RWG^1, and 45% for RWG^2. On the other hand, some of the correctly aligned nodes are lost: 46% and 33% for RWG^1 and RWG^2, respectively. Nonetheless, when choosing alignment candidates, accuracy is more important than recall, given that having many incorrectly aligned pairs of nodes impacts negatively on the performance of $\mathbf{T}'$. To further increase the alignment accuracy, at the expense of recall, we devised a bidirectional best-candidate selection approach by finding out for each imputed G^2, the G^1 node for which it is substituted the most. This way we get a similar best candidate mapping going from G^2 to G^1. Then we only keep the pairings in which both the mappings agree. Table IV shows the results using this selection strategy:

TABLE IV: Alignment with bidirectional selection strategy.

Mode	Alignments	Correct	Acc	Recall
RWG^1	196	173	88.2%	42.7%
RWG^2	135	130	96.2%	58.8%

The alignment accuracy we obtain is considerably high, specially in the case of RWG^2, which points to the fact that our best-candidate selection strategy is successful.

F. Initial RL approach

As discussed previously, our initial approach is to estimate the gradient updates of $\mathbf{F}$ using policy-gradients. Figure 6 shows the $\mathbf{F}$, $\mathbf{T}$, and $\mathbf{D}$ losses during a training session of our RL approach. On the surface, it actually seems to have gone well:

- $\mathbf{F}$ loss drops harshly and then seems to stabilize
- $\mathbf{T}$ loss seems to be dropping steadily
- $\mathbf{D}$ starts high and then seems to stabilize at around 0.6.

However, upon close inspection of the alignment results, we found that $\mathbf{F}$ was suffering from severe mode-collapse. Very early into training, it was imputing the same node over and over again. We believe the model quickly learned that, by imputing the same node, it consistently fooled $\mathbf{D}$ –thus increasing it's reward – using $\mathbf{T}$ to translate such node's embedding to some sort of average of the G^1 embeddings. In short, $\mathbf{G}$ was fooling $\mathbf{D}$ with garbage, which is a common sign of mode collapse.

G. Gumbel-Softmax based approach

To deal with mode collapse, in our second approach we use the WGAN (III-F) loss and update the gradients of $\mathbf{F}$ using Gumbel-Softmax reparametrization (III-E). By freezing $\mathbf{T}'$, we observe the effects of the training on $\mathbf{F}$. Figure 7 shows the behaviour of $\mathbf{F}$ during a training session of 400 iterations when RWG^1 is used. The generator is updated every 5 iterations, as specified in [1].

Figure 7a shows the translation loss. The overall trend of the loss is to increase as training progresses. Furthermore, we observe that, as the $\mathbf{T}$ loss increases, the number of correctly aligned nodes decreases (blue line in Figure 7b, which implies that the translation loss is a good indicator of $\mathbf{F}$'s performance.

Figure 7b also shows the effectiveness of our bidirectional best-candidate selection approach. The dotted red line shows

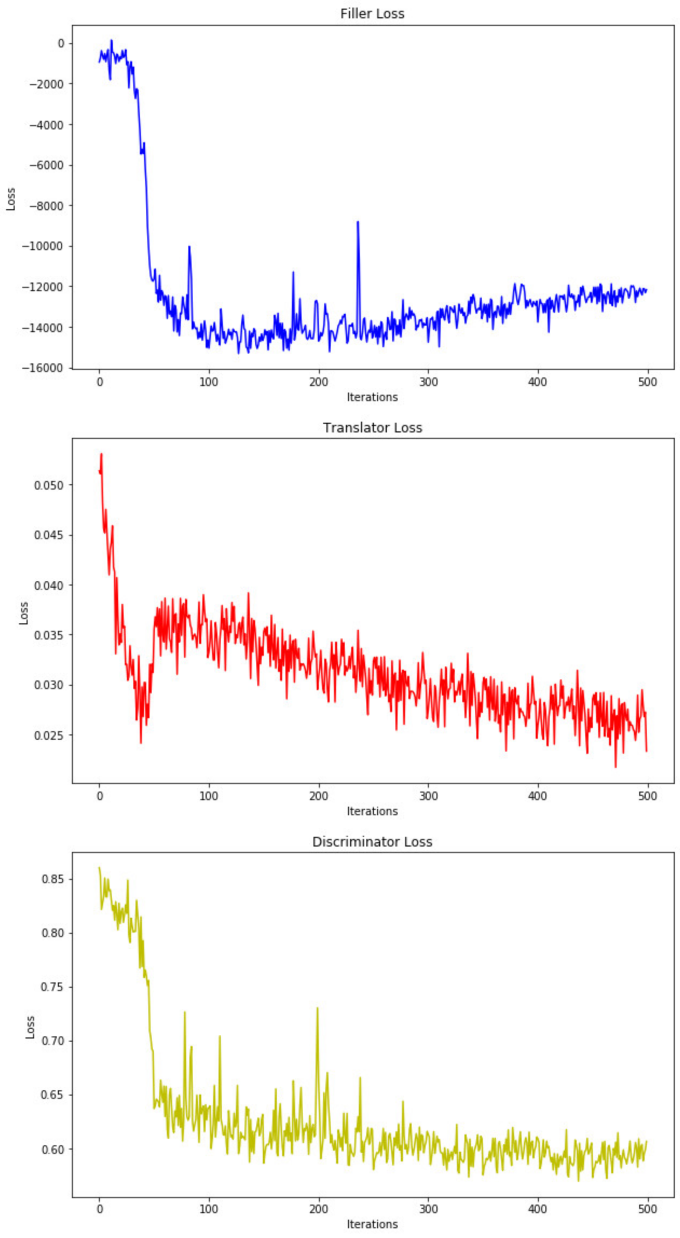


Fig. 6: Training losses of RL approach

the total amount of nodes that have more > 100 appearances in the training data set. The purple line shows how many nodes, out of those with > 100 appearances were initially correctly aligned before starting the training. The orange line represents the amount of node pairings extracted by our selection scheme. Finally, the green line shows how many of these pairings were correctly aligned. Ideally, the orange and green lines would overlap perfectly. This would mean that our selection scheme works perfectly, selecting correctly aligned pairs only. In practice, the orange and green lines are fairly close throughout the whole process, meaning that our selection scheme is doing a good job selecting correctly aligned pairs. It would also be ideal for the orange line to be as close as possible to the purple line, meaning that our approach is able to extract all of the correctly aligned pairs with > 100 appearances. However, during training, the amount of node-pairs extracted by our approach also decreases, meaning that $\mathbf{F}$ is less certain when imputing nodes.

Another relevant effect of the GAN training is shown on Figure 7c: the number of G^2 nodes used to fill the masked walks initially increases before beginning to collapse. The dotted orange line signals the total amount of G^1 nodes encountered in the training dataset, thus, once the blue line is below it, the model is effectively collapsing.

Finally, Figure 7d presents the accuracy and recall of our selection strategy. It's note-worthy that recall remains stable during the initial imputing diversification. Accuracy, however, has a clear decreasing trend throughout .

Similarly, Figure 8 shows $\mathbf{F}$s performance when using the restricted random-walk generation process RWG^2. Some of the observations made for RWG^1 still hold: the translator loss is a good indicative of $\mathbf{F}$'s performance and the aligned pairs decrease with training. There are, however, some noticeable differences. The model does not seem to reach mode collapse, for example. Additionally, due to the restricted way random-walks are created, less G^1 nodes are encountered which makes it easier for our selection approach to correctly select aligned pairs (Figure 8b). In any case, training does not seem to be effective in this case either.

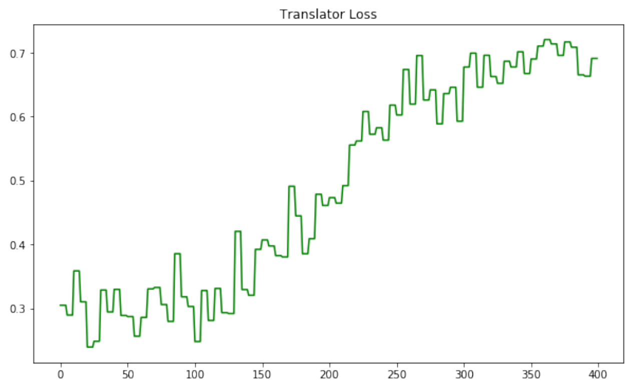
GAN training seems to be beneficial for the first few iterations: the initial increase in node diversity helps with mode collapse without decreasing the number of correctly aligned nodes. However, the ability of our strategy to select correctly aligned nodes quickly starts decreasing after a few iterations, as the orange and green lines start to diverge before taking a steep decline that seems to align with the model starting to collapse. While this phenomenon seems to be consistent throughout our experiments, due to distinct weight initialization and randomized training, its behaviour is slightly different in each training session. As such, we have not been able to determine a suitable stopping point that would generalize to all cases.

H. *Additional experiments*

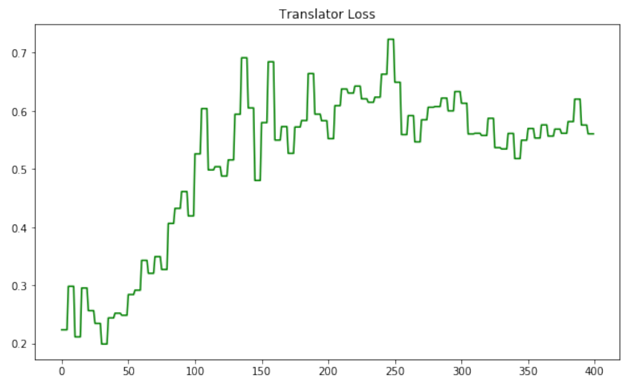
We have yet to find a model configuration in which GAN training proves beneficial in the long run. We have performed extensive experimentation with hyper-parameters and other configuration points such as:

- Attention modules for the decoders in $\mathbf{F}$ and $\mathbf{D}$.
- WGAN with Gradient Penalty.
- Standard GAN loss with Gumbel-Softmax.
- Random-walk length and curriculum training.
- Embedding generation techniques.
- Embedding and hidden state dimensionality.

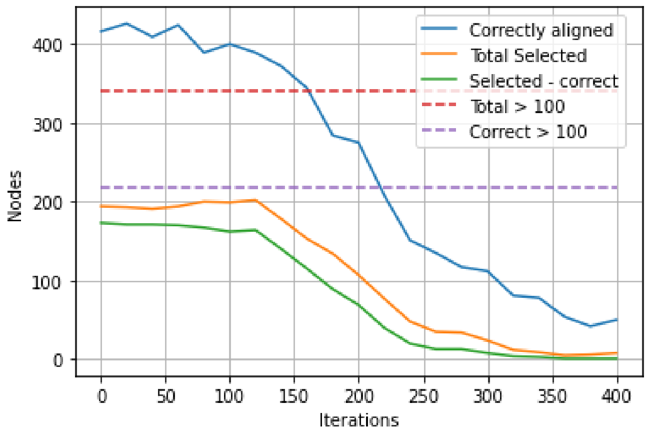
Examples of these experiments can be found in Appendix A. All of them, however, follow a similar pattern to the one discussed previously in this section. At this point, we believe the issue goes beyond hyper-parameter tuning.



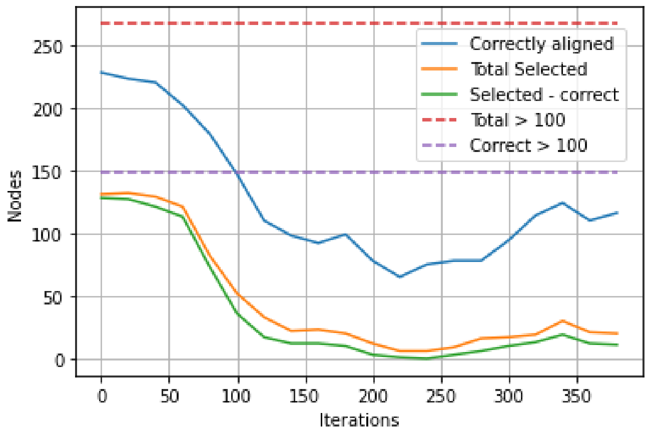
(a) Translator Loss.



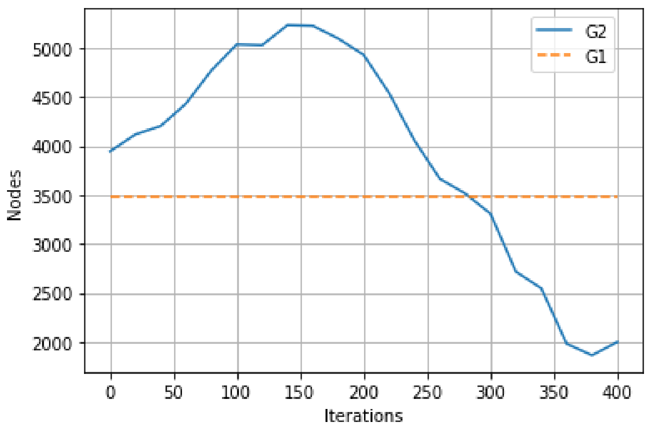
(a) Translator Loss.



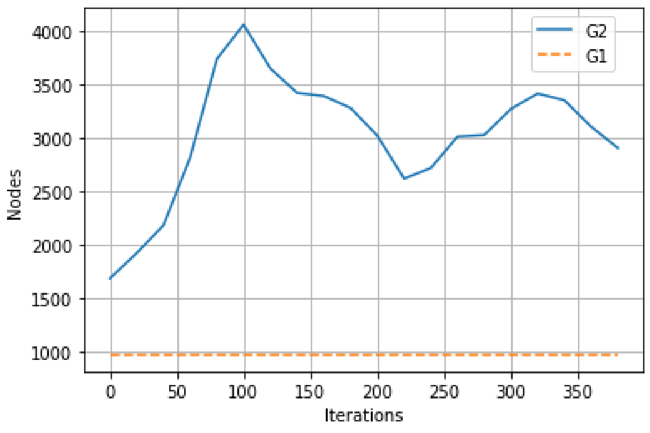
(b) Node alignment.



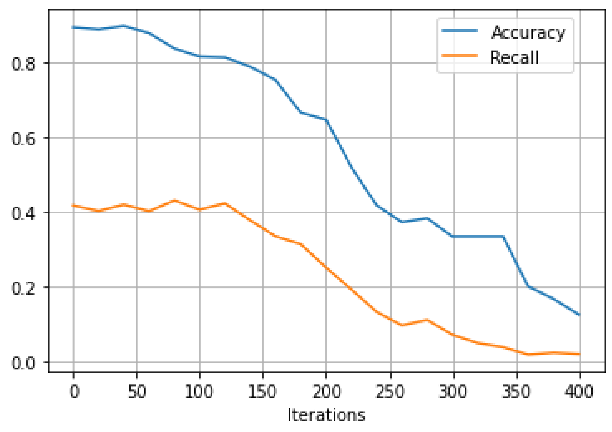
(b) Node alignment.



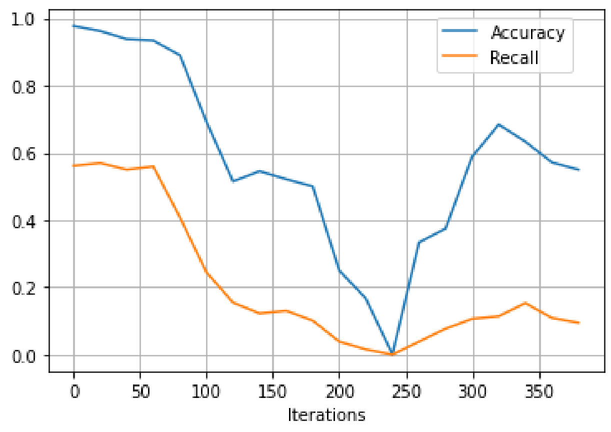
(c) Fill in node usage.



(c) Fill in node usage.



(d) Accuracy and Recall.



(d) Accuracy and Recall.

Fig. 7: **F** training behaviour with RWG[1].

Fig. 8: **F** training behaviour with RWG[2].

I. Non-GAN alternative

From our current results, we devise the following iterative alignment process that does away with the GAN framework:

Algorithm 3 Iterative alignment by Random-Walk imputing

```
1:  procedure NON_GAN_ALIGN($G^1$,$G^2$,$Anc$)
2:      Pre-train $\mathbf{F}$
3:      do
4:          $W^1 \leftarrow$ RWG($Anc$)
5:          $W^2 \leftarrow$ Mask($W^1$)
6:          $\hat{W}^2 \leftarrow \mathbf{F}(W^2)$
7:          $Selected \leftarrow$ best_candidates($W^1, \hat{W}^2$)
8:              $Anc \leftarrow Anc \cup Selected$
9:      while $|Selected| > 0$
10:     Train $\mathbf{T}$
11:     return $\mathbf{T}$
12: end procedure
```

The process simply uses a pre-trained $\mathbf{F}$ to impute the masked walks W^2, and iteratively augments the initial set of anchors Anc using our bidirectional best-candidate selection strategy until no further alignments are found. However, we have not tested its performance given that it completely removes any GAN-based training which defeats the main motivation of our work.

V. Conclusions

Even though we have yet to be successful in our GAN training experiments, we still believe there is value in our approach to NA because of its novelty in using $\mathbf{F}$ to fill-in masked random walks. This, supported by the alignment results achieved by our candidate selection strategy that our experiments have shown to be effective at selecting correctly matched pairs.

We have some ideas on how to continue working on solving our current shortcomings such as creating a custom loss function that incorporates supervised information to help guide the GAN training procedure, and improving the fill-in performance of $\mathbf{F}$ by providing it with additional information such as explicit mask embeddings or positional ones.

At the same time, we are aware that a simple fix might not be easily found and a more comprehensive revision of our framework could be necessary.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. 70:214–223, 06–11 Aug 2017.

[2] M. Bayati, M. Gerritsen, D. F. Gleich, A. Saberi, and Y. Wang. Algorithms for large, sparse network alignment problems. pages 705–710, 2009.

[3] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann. NetGAN: Generating graphs via random walks. 80:610–619, 10–15 Jul 2018.

[4] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. pages 891–900, 2015.

[5] L. Chaozhuo, W. Senzhang, Y. Philip, L. Yanbo, L. Yun, and L. Zhoujun. Snna: Adversarial learning for weakly-supervised social network alignment. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33, 2019.

[6] X. Chu, X. Fan, D. Yao, Z. Zhu, J. Huang, and J. Bi. Cross-network embedding for multi-network alignment. pages 273–284, 2019.

[7] W. Fedus, I. Goodfellow, and A. Dai. Maskgan: Better text generation via filling in the ____. *ICML 2018*, 2018.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial networks. *CoRR*, abs/1406.2661, 2014.

[9] A. Grover and J. Leskovec. Node2vec: Scalable feature learning for networks. pages 855–864, 2016.

[10] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[11] M. Heimann, H. Shen, T. Safavi, and D. Koutra. Regal: Representation learning-based graph alignment. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM)*, 2018.

[12] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *ICLR 2018*, 2017.

[13] D. Kiela, E. Grave, A. Joulin, and T. Mikolov. Efficient large-scale multi-modal classification. pages 5198–5204, 2018.

[14] D. Koutra, H. Tong, and D. Lubensky. Big-align: Fast bipartite graph alignment. *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 389–398, 12 2013.

[15] L. Liu, W. K. Cheung, X. Li, and L. Liao. Aligning users across social networks using network embedding. *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*, pages 1774–1780, 2016.

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[17] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. *In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, abs/1403.6652, 2014.

[18] O. Press, A. Bar, B. Bogin, J. Berant, and L. Wolf. Language generation with recurrent generative adversarial networks without pre-training. *CoRR*, abs/1706.01399, 2017.

[19] S. Rajeswar, S. Subramanian, F. Dutil, C. J. Pal, and A. C. Courville. Adversarial generation of natural language. *CoRR*, abs/1705.10929, 2017.

[20] S. Richard, M. David, S. Satinder, and M. Yisha. Policy gradient methods for reinforcement learning with

function approximation. *Advances in neural information processing systems*, page 1057–1063, 2000.

[21] R. Singh, J. Xu, and B. Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proceedings of the National Academy of Sciences*, 10(35):12763–12768, 2008.

[22] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: large-scale information network embedding. *In Proceedings of the 24th International Conference on World Wide Web*, abs/1503.03578, 2015.

[23] S. Wang, X. Li, Y. Ye, S. Feng, R. Y. K. Lau, X. Huang, and X. Du. Anchor link prediction across attributed networks via network embedding. *Entropy*, 21(3), 2019.

[24] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016.

[25] J. Zhang, B. Chen, X. Wang, H. Chen, C. Li, F. Jin, G. Song, and Y. Zhang. Mego2vec: Embedding matched ego networks for user alignment across social networks. In *ACM International Conference on Information and Knowledge Management*, pages 327–336, 2018.

[26] S. Zhang and H. Tong. Final: Fast attributed network alignment. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*, pages 1345–1354, 2016.

[27] Y. Zhang, Z. Gan, and L. Carin. Generating text via adversarial training. *NIPS*, 2016.

[28] F. Zhou, L. Liu, K. Zhang, G. Trajcevski, J. Wu, and T. Zhong. Deeplink: A deep learning approach for user identity linkage. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1313–1321, 2018.

Appendix

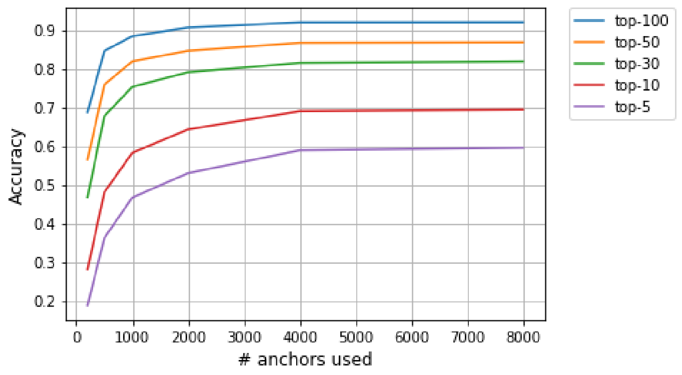


Fig. 9: *Top-k* alignment accuracy for a $\mathbf{T}'$ with 2 hidden layers.

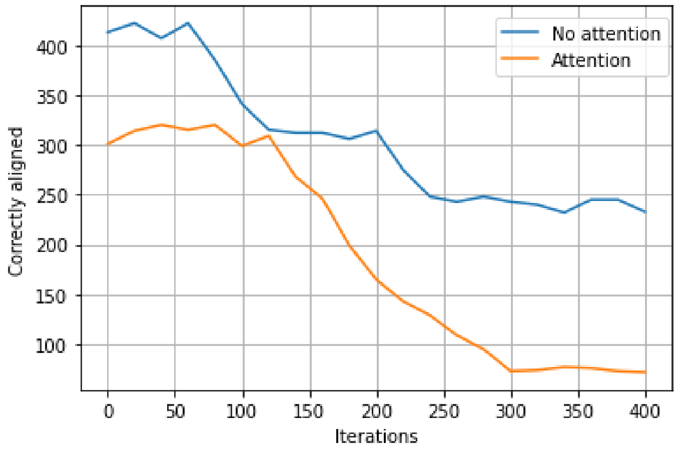


Fig. 10: Decoders with Attention modules.

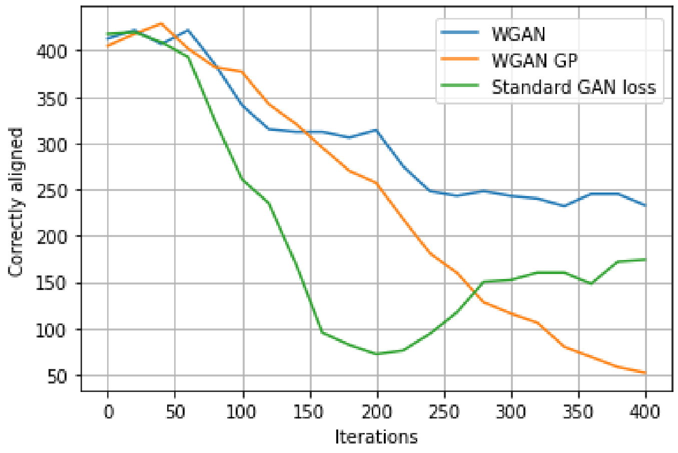


Fig. 11: Different loss functions.

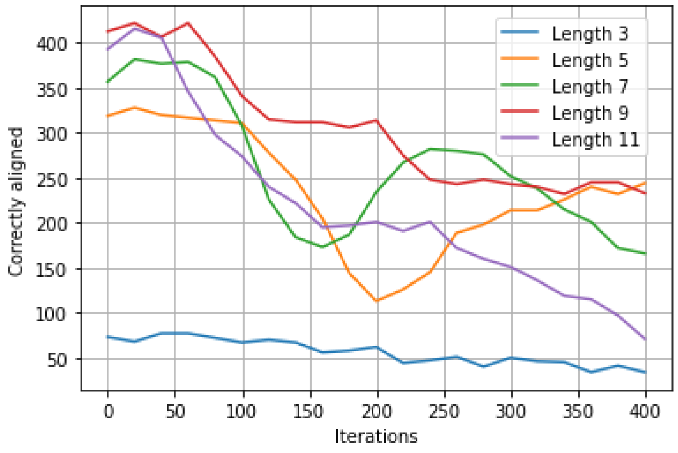


Fig. 12: Different random-walk lengths.

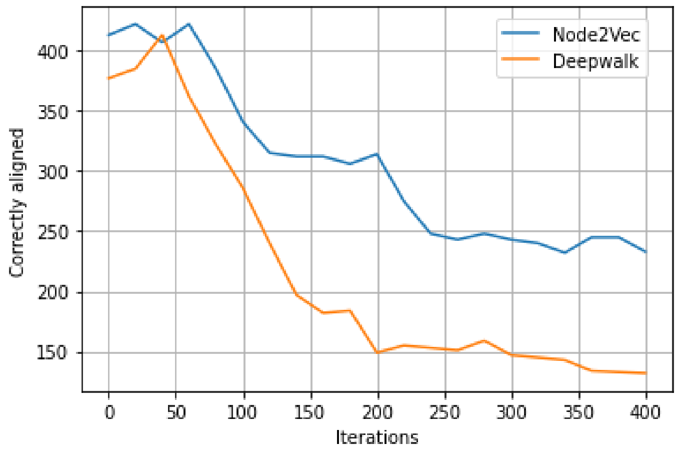


Fig. 13: Different embedding techniques.

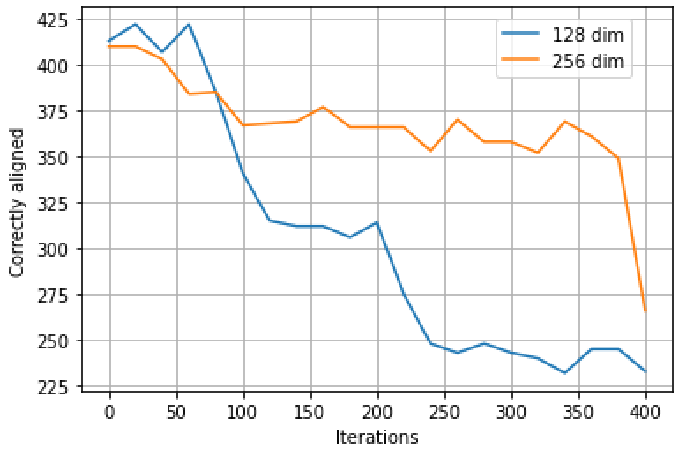


Fig. 14: Different embedding dimensions.