Within REGAL we devise xNetMF, an elegant and principled node embedding formulation that uniquely generalizes to multi-network problems. Our results demonstrate the utility and promise of unsupervised representation learning-based network alignment in terms of both speed and accuracy. REGAL runs up to 30× faster in the representation learning stage than comparable methods, outperforms existing network alignment methods by 20 to 30% accuracy on average, and scales to networks with millions of nodes each.

**CCS CONCEPTS**
- Information systems → Data mining
- Computing methodologies → Learning latent representations;

**KEYWORDS**
graph mining, network alignment, graph matching, node representation learning, node embedding

**ACM Reference Format:**

**1 INTRODUCTION**

Networks are powerful structures that naturally capture the wealth of relationships in our interconnected world, such as co-authorships, email exchanges, and friendships [17]. The data mining community has accordingly proposed various methods for numerous tasks over a single network, like anomaly detection, link prediction, and user modeling. However, many graph mining tasks involve joint analysis of nodes across multiple networks. Some problems, like network alignment [2, 18, 42] and graph similarity [19], are inherently defined in terms of multiple graphs. In other cases, it is desirable to perform analysis across a collection of graphs, such as the MRI-based brain graphs of patients [7], or snapshots of a temporal graph [33].

In this work, we study network alignment or matching, which is the problem of finding corresponding nodes in different networks. Network alignment is crucial for identifying similar users in different social networks, analyzing chemical compounds, studying protein-protein interaction, and various computer vision tasks, among others [2]. Many existing methods try to relax the computationally hard optimization problem, as designing features that can be directly compared for nodes in different networks is not an easy task. However, recent advances [9, 28, 35, 39] have automated the process of learning node feature representations and have led to state-of-the-art performance in downstream prediction, classification, and clustering tasks. Motivated by these successes, we propose network alignment via matching latent, learned node representations. Formally, the problem can be stated as:

**Problem 1.** Given two graphs $G_1$ and $G_2$ with node-sets $V_1$ and $V_2$ and possibly node attributes $A_1$ and $A_2$ resp., devise an efficient network alignment method that aligns nodes by learning directly comparable node representations $Y_1$ and $Y_2$, from which a node mapping $\phi : V_1 \rightarrow V_2$ between the networks can be inferred.

To this end, we introduce REGAL, or REpresentation-based Graph Alignment, a framework that efficiently identifies node
Problem Formulation. We formulate the important unsupervised graph alignment problem as a problem of learning and matching node representations that generalize to multiple graphs. To the best of our knowledge, we are the first to do so.

Principled Algorithms. We introduce a flexible alignment framework, REGAL (Fig. 1), which learns node alignments by jointly embedding multiple graphs and comparing the most similar embeddings across graphs without performing all pairwise comparisons. Within REGAL we devise $x$NetMF, an elegant and principled representation learning formulation. $x$NetMF learns embeddings from structural and, if available, attribute identity, which are characteristics most conducive to multi-network analysis.

Extensive Experiments. Our results demonstrate the utility of representation learning-based network alignment in terms of both speed and accuracy. Experiments on real graphs show that $x$NetMF runs up to 30× faster than several existing network embedding techniques, and REGAL outperforms traditional network alignment methods by 20-30% in accuracy.

For reproducibility, the source code of REGAL and $x$NetMF is publicly available at https://github.com/GemsLab/REGAL.

2 RELATED WORK

Our work focuses on the problem of network alignment, and is related to node representation learning and matrix approximation.

Network Alignment. Instances of the network alignment or matching problem appear in various settings: from data mining to security and re-identification [2, 18, 42], chemistry, bioinformatics [16, 34, 36], databases, translation [2], vision, and pattern recognition [41]. Network alignment is usually formulated as the optimization problem $\min_{P} \| PA_1 P^T - A_2 \|_F^2$ [18], where $A_1$ and $A_2$ are the adjacency matrices of the two networks to be aligned, and $P$ is a permutation matrix or a relaxed version thereof, such as doubly stochastic matrix [38] or some other concave/convex relaxation [41]. Popular proposed solutions to the network alignment problem span genetic algorithms, spectral methods, clustering algorithms, decision trees, expectation maximization, probabilistic approaches, and distributed belief propagation [2, 16, 34, 36]. These methods usually require carefully tailoring for special formats or properties of the input graphs. For instance, specialized formulations may be used when the graphs are bipartite [18] or contain node/edge attributes [42], or when some “seed” alignments are known a priori [15]. Prior work using node embeddings designed for social networks to align users [26] has required such seed alignments. In contrast, our approach can be applied to attributed and unattributed graphs with virtually no change in formulation, and is unsupervised: it does not require prior alignment information to find high-quality matchings. Recent work [12] has used hand-engineered features, while our proposed approach leverages the power of latent feature representations.

Node Representation Learning. Representation learning methods try to find similar embeddings for similar nodes [8]. They may be based on shallow [9] or deep architectures [39], and may discern neighborhood structure through random walks [28] or first- and second-order connections [35]. Recent work inductively learns representations [10] and/or incorporates textual or other node attributes [14, 40]. However, all these methods use node proximity or neighborhood overlap to drive embedding, which has been shown to lead to inconsistency across networks [11].

Unlike these methods, the recent work struc2vec [31] preserves structural similarity of nodes, regardless of their proximity in the network. Prior to this work, existing methods for structural role discovery mainly focused on hand-engineered features [32]. However, for structurally similar nodes, struc2vec embeddings were found to be visually more comparable [31] than those learned by state-of-the-art proximity-based node embedding techniques as well as existing methods for role discovery [13]. While this work is most closely related to our proposed node embedding method, we summarize some crucial differences in Table 1. Additionally, we note that struc2vec, like work on structural node embeddings concurrent to ours [5], cannot natively use node attributes.

Table 1: Qualitative comparison of related work to the embedding module of REGAL. (*: Method not based on random walks, RW)

<table>
<thead>
<tr>
<th></th>
<th>Structure</th>
<th>Attributes</th>
<th>RW-free*</th>
<th>Scalable</th>
<th>Cross-net</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINE [35]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>TADW [40]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>node2vec [9]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>struc2vec [31]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>xNetMF (in REGAL)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2: Qualitative comparison of structure-based embeddings.

<table>
<thead>
<tr>
<th></th>
<th>Variable-length degree sequences compared with dynamic time warping</th>
<th>Fixed length vectors capturing neighborhood degree distributions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Variance-inducing, time-consuming random walk-based sampling</td>
<td>Efficient matrix factorization</td>
</tr>
<tr>
<td></td>
<td>Heuristic-based omission of similarity computations</td>
<td>Low-rank implicit approximation of full similarity matrix</td>
</tr>
<tr>
<td>Node</td>
<td>Computation time for network embeddings:</td>
<td>(Table 5) [4] using optimizations</td>
</tr>
<tr>
<td>Similarity</td>
<td>∼0.5 hours to embed Arxiv network</td>
<td>~&lt;90 sec to embed Arxiv network; ~22X speedup</td>
</tr>
</tbody>
</table>

Note: For reproducibility, the source code of REGAL and $x$NetMF is publicly available at https://github.com/GemsLab/REGAL.
Many well-known node embedding methods based on shallow architectures such as the popular skip-gram with negative sampling (SGNS) have been cast in matrix factorization frameworks \cite{30, 40}. However, ours is the first to cast node embedding using SGNS to capture structural identity in such a framework. In Table 2 we extend our qualitative comparison to some other well-known methods that use similar architectures. Their limitations inspire many of our choices in the design of REGAL and xNetMF.

In terms of applications, very few works consider using learned representations for problems that are inherently defined in terms of multiple networks, where embeddings must be compared. \cite{27} computes a similarity measure between graphs based on the Earth Mover’s Distance \cite{23} between simple node embeddings generated from the eigendecomposition of the adjacency matrix. Here, we consider the significantly harder problem of learning embeddings that may be individually matched to infer node-level alignments.

### Low-Rank Matrix Approximation

The Nyström method has been used for low-rank approximations of large, dense similarity matrices \cite{6}. While the quality of its approximation has been extensively studied theoretically and empirically in a statistical learning context for kernel machines \cite{1}, to the best of our knowledge it has not been considered in the context of node embedding.

### 3 REGAL: REPRESENTATION

### LEARNING-BASED GRAPH ALIGNMENT

In this section we introduce our representation learning-based network alignment framework, REGAL, for Problem 1. For simplicity we focus on aligning two graphs (e.g., social or protein networks), though our method can easily be extended to more networks. Let $G_1(\mathcal{V}_1, \mathcal{E}_1)$ and $G_2(\mathcal{V}_2, \mathcal{E}_2)$ be two unweighted and undirected graphs with node sets $\mathcal{V}_1$ and $\mathcal{V}_2$; edge sets $\mathcal{E}_1$ and $\mathcal{E}_2$; and possibly node attributes $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Note that these graphs do not have to be the same size, unlike many other network alignment formulations that have this (often unrealistic) restriction. Let $n$ be the number of nodes across graphs, i.e., $n = |\mathcal{V}_1| + |\mathcal{V}_2|$. We define the main symbols in Table 3.

The steps of REGAL may be summarized as:

1. **Node Identity Extraction**: The first step extracts structure- and attribute-related information for all $n$ nodes.

2. **Efficient Similarity-based Representation**: The second step obtains the node embeddings, conceptually by factorizing a similarity matrix of the node identities from the previous step. To avoid the expensive computation of pairwise node similarities and explicit factorization, we extend the Nyström method for low-rank matrix approximation to perform an implicit similarity matrix factorization by (a) comparing the similarity of each node only to a sample of $p \ll n$ “landmark” nodes, and (b) using these node-to-landmark similarities to construct our representations from a decomposition of its low-rank approximation.

3. **Fast Node Representation Alignment**: Finally, we align nodes between graphs by greedily matching the embeddings with an efficient data structure that allows for fast identification of the top-$\alpha$ most similar embeddings from the other graph(s).

In the rest of this section we discuss and justify each step of REGAL, the pseudocode of which is given in Algorithm 1. Note that the first two steps, which output a set of node embeddings, comprise our xNetMF method, which may be independently used, particularly for further cross-network analysis tasks.

### 3.1 Step 1: Node Identity Extraction

The goal of REGAL’s representation learning module, xNetMF, is to define node “identity” in a way that generalizes to multi-network problems. This step is critical because many existing works define identity based on node-to-node proximity, but in multi-network problems nodes have no direct connections to each other and thus cannot be sampled in each other’s contexts by random walks on separate graphs. To overcome this problem, we focus instead on more broadly comparable, generalizable quantities: structural identity, which relates to structural roles \cite{13}, and attribute-based identity.

#### Structural Identity

In network alignment, the well-established assumption is that aligned nodes have similar structural connectivity or degrees \cite{18, 42}. Adhering to this assumption, we propose to learn about a node’s structural identity from the degrees of its neighbors. To gain higher-order information, we also consider neighbors up to $k$ hops from the original node.

For a node $u \in \mathcal{V}$, we denote $\mathcal{R}_u^k$ as the set of nodes that are exactly $k \geq 0$ steps away from it in its graph $G_i$. We want to capture degree information about the nodes in $\mathcal{R}_u^k$. A basic approach would be to store the degrees in a $D$-dimensional vector $d_u$, where $D$ is the maximum degree in the original graph $G$, with the $i$-th entry of $d_u$, or $d_u(i)$, the number of nodes in $\mathcal{R}_u^k$ with degree $i$. For simplicity, an example of this approach is shown for the vectors $d_A, d_B$, etc. in Fig. 2. However, real graphs have skewed degree distributions. To prevent one high-degree node from inflating the length of these vectors, we bin nodes together into $b = \lceil \log_2 D \rceil$ logarithmically scaled buckets such that the $i$-th entry of $d_u^b$ contains the number of nodes $u \in \mathcal{R}_u^k$ such that $\{\log_2(\text{deg}(u))\} = i$. This has two benefits: (1) it shortens the vectors $d_u^b$ to a manageable $\lceil \log_2 D \rceil$ dimensions, and (2) it makes their entries more robust to small changes in degree introduced by noise, especially for high degrees when more different degree values are combined into one bucket.

#### Attribute-Based Identity

Node attributes, or features, have been shown to be useful for cross-network tasks \cite{42}. Given $F$ node attributes, we can create for each node $u$ an $F$-dimensional vector

---

**Table 3: Major symbols and definitions.**

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_i(\mathcal{V}_i, \mathcal{E}_i, \mathcal{A}_i)$</td>
<td>graph $i$ with node set $\mathcal{V}_i$, edge set $\mathcal{E}_i$, and node attributes $\mathcal{A}_i$</td>
</tr>
<tr>
<td>$n$</td>
<td>number of nodes in graph $G_i$</td>
</tr>
<tr>
<td>$\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$</td>
<td>combined set of vertices in $G_1$ and $G_2$</td>
</tr>
<tr>
<td>$</td>
<td>\mathcal{V}</td>
</tr>
<tr>
<td>$d^{\text{avg}}$</td>
<td>average node degree</td>
</tr>
<tr>
<td>$\mathcal{R}_u^k$</td>
<td>set of $k$-hop neighbors of node $u$</td>
</tr>
<tr>
<td>$d_u$</td>
<td>vector of node degrees in a single set $\mathcal{R}_u^k$</td>
</tr>
<tr>
<td>$K'$</td>
<td>maximum hop distance considered</td>
</tr>
<tr>
<td>$\delta$</td>
<td>discount factor in $(0, 1]$ for distant neighbors</td>
</tr>
<tr>
<td>$d_u^\delta$</td>
<td>combined neighbor degree vector for node $u$</td>
</tr>
<tr>
<td>$b$</td>
<td>number of buckets for degree binning</td>
</tr>
<tr>
<td>$\mathcal{I}_u$</td>
<td>$F$-dimensional attribute vector for node $u$</td>
</tr>
<tr>
<td>$\mathcal{S}, \mathcal{S}$</td>
<td>combined structural and attribute-based similarity matrix, and its approximation</td>
</tr>
<tr>
<td>$\mathcal{Y}$</td>
<td>matrix with node embeddings as rows, and its approximation</td>
</tr>
<tr>
<td>$\rho$</td>
<td>number of landmark nodes in REGAL</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>the number of alignments to find per node</td>
</tr>
</tbody>
</table>
we note that statistics such as the mean or standard deviation of
Thus, we compare structural identity at several levels by combining
The discount factor is set to $\delta = 0.5$. For simplicity, no logarithmic binning is applied on $d^n_k$.

$f_u$ representing its values (or lack thereof). For example, $f_u(i)$
corresponds to the $i^{th}$ attribute value for node $u$. Since we focus on
node representations, we mainly consider node attributes, although
we note that statistics such as the mean or standard deviation of
dge attributes on incident edges to a node can easily be turned
into node attributes. Note that while REGAL is flexible to incor-
porate attributes, if available, it can also rely solely on structural
information when such side information is not available.

Cross-Network Node Similarity. We now incorporate the above
aspects of node identity into a combined similarity function that
can be used to compare nodes within or across graphs, relying on
the comparable notions of structural and attribute identity, rather
than direct proximity of any kind:

$$
sim(u, v) = \exp \left(-\gamma_s \cdot ||d_u - d_v||^2 - \gamma_a \cdot \text{dist}(f_u, f_v)\right),
$$

where $\gamma_s$ and $\gamma_a$ are scalar parameters controlling the effect of the structural and attribute-based identity respectively; \text{dist}(f_u, f_v)$ is the attribute-based distance of nodes $u$ and $v$, discussed below (this term is ignored if there are no attributes); $d_u = \sum_{k=1}^{K} \delta^{k-1}d^n_k$ is the neighbor degree vector for node $u$ aggregated over $K$ different hops; $\delta \in (0, 1]$ is a discount factor for greater hop distances; and $K$ is a maximum hop distance to consider (up to the graph diameter). Thus, we compare structural identity at several levels by combining the neighborhood degree distributions at several hop distances, attenuating the influence of distant neighborhoods with a weighting schema that is often encountered in diffusion processes [19].

The distance between attribute vectors depends on the type of
node attributes (e.g., categorical, real-valued). A variety of functions
can be employed accordingly. For categorical attributes, which have
been studied in attributed network alignment [42], we propose us-
ing the number of disagreeing features as a attribute-based distance
measure of nodes $u$ and $v$: \text{dist}(f_u, f_v) = \sum_{f \in F} I_{f(i)\neq f(i)}$, where $I$ is the indicator function. Real-valued attributes can be compared by Euclidean or cosine distance, for example.

3.2 Step 2: Efficient Similarity-based Representation
As we have mentioned, many representation learning methods are
stochastic [9, 28, 31, 35, 39]. A subset of these rely on random walks
on the original graph [9, 28] or a generated multi-layer similarity
graph [31]) to sample context for the SGNS embedding model. For
cross-network analysis, we avoid random walks for two reasons:
(1) The variance they introduce in the representation learning often
makes embeddings across different networks non-comparable [11];
(2) they can add to the computational expense. For example, node2vec’s total runtime is dominated by its sampling time [9].

To overcome the aforementioned issues, we propose a new im-
plicit matrix factorization-based approach that leverages a com-
bined structural and attribute-based similarity matrix $S$, which is
induced by our similarity function in Eq. (1) and considers affinities
different neighborhoods. Intuitively, the goal is to find $n \times p$ ma-
trices $Y$ and $Z$ such that: $S \approx YZ^\top$, where $Y$ is the node embedding
matrix and $Z$ is not needed for our purposes. We first discuss the
limitations of traditional approaches, then propose an efficient way
of obtaining the embeddings without ever explicitly computing $S$.

Limitations of Existing Approaches. A natural but naive
approach is to compute combined structural and attribute-based sim-
ilarities between all pairs of nodes within and across both graphs to
form the matrix $S$, such that $S_{ij} = \text{sim}(i, j) \forall i, j \in V$. Then $S$ can be
explicitly factorized, for example by minimizing a factorization loss
function given $S$ as input, (e.g., the Frobenius norm $||S - YZ^\top||^2_F$
[21]). However, both the computation and storage of $S$ have qua-
dratic complexity in $n$. While this would allow us to embed graphs
jointly, it lacks the needed scalability for multiple large networks.

Another alternative is to create a sparse similarity matrix by
calculating only the “most important” similarities, for each node
choosing a small number of comparisons using heuristics like simi-
arity of node degree [31]. However, such ad-hoc heuristics may be
fragile in the context of noise. We will have no approximation at
all for most of the similarities, and there is no guarantee that the
most important ones are computed.

Step 2a: Reduced $n \times p$ Similarity Computation. Instead, we
propose a principled way of approximating the full similarity matrix
$S$ with a low-rank matrix $\tilde{S}$, which is never explicitly computed.
To do so, we randomly select $p < n$ “landmark” nodes chosen
across both graphs $G_1$ and $G_2$ and compute their similarities to all
$n$ nodes in these graphs using Eq. (1). This yields an $n \times p$ similarity
matrix $C$, from which we can extract a $p \times p$ “landmark-to-landmark”
submatrix $W$. As we explain below, these two matrices suffice to
approximate the full similarity matrix and allow us to obtain node
embeddings without actually computing and factorizing $S$.
To do so, we extend the Nyström method, which has applications in randomized matrix methods for kernel machines [6], to node embedding. The low-rank matrix $\tilde{S}$ is given as:

$$
\tilde{S} = CW^T C^T,
$$

where $C$ is an $n \times p$ matrix formed by sampling $p$ landmark nodes from $V$ and computing the similarity of all $n$ nodes of $G_1$ and $G_2$ to the $p$ landmarks only, as shown in Fig. 2. Meanwhile, $W^T$ is the pseudoinverse of $W$, a $p \times p$ matrix consisting of the pairwise similarities among the landmark nodes (it corresponds to a subset of $p$ rows of $C$). We choose landmarks randomly; more elaborate (and slower) sampling techniques based on leverage scores [1] or node centrality measures offer little, if any, performance improvement.

Because $S$ contains an estimate for the similarity between any pair of nodes in either graph, it would still take $\Omega(n^2)$ time and space to compute and store. However, as we discuss below, to learn node representations we never have to explicitly construct $S$ either.

**Step 2b: From Similarity to Representation.** Recall that our ultimate interest is not in the similarity matrix $S$ or even an approximation such as $\tilde{S}$, but in the node embeddings that we can obtain from a factorization of the latter. We now show that we can actually obtain these from the decomposition in Eq. (2).

**Theorem 3.1.** Given graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ with $n \times n$ joint combined structural and attribute-based similarity matrix $S \approx YZ^T$, its node embedding matrix $Y$ can be approximated as

$$
Y = CU\Sigma^{1/2},
$$

where $C$ is the $n \times p$ matrix of similarities between the $n$ nodes and $p$ randomly chosen landmark nodes, and $W^T = \Sigma \Upsilon^T$ is the full rank singular value decomposition of the pseudoinverse of the small $p \times p$ landmark-to-landmark similarity matrix $W$.

**Proof.** Given the full-rank SVD of the $p \times p$ matrix $W^T$ as $U \Sigma \Upsilon^T$, we can rewrite Eq. (2) as $S \approx \tilde{S} = C(U \Sigma \Upsilon^T)C^T = (CU\Sigma^{1/2})^T \Sigma^{1/2} V^T C^T = YZ^T$.

Now, we never have to construct an $n \times n$ matrix and then factorize it (i.e., by optimizing a nonconvex factorization objective). Instead, to derive $Y$, the only node comparisons we need are for the $n \times p$ "skinny" matrix $C$, while the expensive SVD is performed only on its small submatrix $W$. Thus, we can obtain node representations by implicitly factorizing $\tilde{S}$, a low-rank approximation of the full similarity matrix $S$. The $p$-dimensional node embeddings of the two input graphs $G_1$ and $G_2$ are then subsets of $Y$: $Y_1$ and $Y_2$, respectively. This construction corresponds to the explicit factorization (Fig. 3), but at significant runtime and storage savings.

---

**Algorithm 1 REGAL ($G_1, G_2, p, K, y_f, y_a, \alpha$)**

1. **Step 1.** Learn embedding representations $\tilde{Y}_1, \tilde{Y}_2$ for node pairs $\{u, v\}$ using $\text{NetMF}$ or $\text{struc2vec}$
2. **Step 2a.** Build dense $p \times p$ similarity graphs $G_1, G_2$
3. **Step 2b.** From similarity to representation
4. **Step 3.** Fast Node Representation Alignment

---

**Algorithm 2 xNetMF ($G_1, G_2, p, K, y_f, y_a$)**

1. **Step 1.** Node Identity Extraction
2. **Step 2a.** Reduced $n \times p$ similarity computation
3. **Step 2b.** From similarity to representation
4. **Step 3.** Fast Node representation alignment

As stated earlier, xNetMF, which we summarize in Alg. 2, forms the first two steps of REGAL. The postprocessing step, where we normalize the magnitude of the embeddings, makes them more comparable based on Euclidean distance, which we use in REGAL.

**Connection between xNetMF and SGNs.** We show a formal connection between matrix factorization, the technique behind our xNetMF, and a variant of the struc2vec framework: another form of structure-based embedding optimized with SGNs [31] in Appendix A. Indeed, similar equivalences between SGNs and matrix factorization have been studied [24, 25] and applied to proximity-based node embedding methods [30], but ours is the first to explore such connections for methods that preserve structural identity.

**3.3 Step 3: Fast Node Representation Alignment**

The final step of REGAL is to efficiently align nodes using their representations, assuming that two nodes $u \in V_1$ and $v \in V_2$ may match if their xNetMF embeddings are similar. Let $Y_1$ and $Y_2$ be matrices of the $p$-dimensional embeddings for nodes in graphs $G_1$ and $G_2$. We take the likelihood of (soft) alignment to be proportional
to the similarity between the nodes' embeddings. Thus, we greedily align nodes to their closest match in the other graph based on embedding similarity, as shown in Fig. 2. This method is simpler and faster than optimization-based approaches, and works thanks to high-quality node feature representations.

Data structures for efficient alignment. A natural way to find the alignments for each node is to compute all pairs of similarities between node embeddings (i.e., the rows of \( \tilde{Y}_1 \) and \( \tilde{Y}_2 \)) and choose the top-1 for each node. Of course, this is not desirable due to its inefficiency. Since in practice only the top-n most likely alignments are used, we turn to specialized data structures for quickly finding the closest data points. We store the embeddings \( \tilde{Y}_2 \) in a k-d tree, a data structure used to accelerate exact similarity search for nearest neighbor algorithms and many other applications [3].

For each node in \( G_1 \), we can quickly query this tree with its embedding to find the \( \alpha \ll n \) closest embeddings from nodes in \( G_2 \). This allows us to compute "soft" alignments for each node by returning one or more nodes in the opposite graph with the most similar embeddings, unlike many existing alignment methods that only find "hard" alignments [2, 16, 34, 42]. Here, we define the similarity between the \( p \)-dimensional embeddings of nodes \( u \) and \( v \) as \( \text{sim}_{emb}(\tilde{Y}_1[u], \tilde{Y}_2[v]) = e^{-||Y_1[u] - Y_2[v]||_2^2} \), which converts the Euclidean distance to similarity. Since we only want to align nodes to counterparts in the other graph, we only compare embeddings in \( \tilde{Y}_1 \) with ones in \( \tilde{Y}_2 \). If multiple top alignments are desired, they may be returned in sorted order by their embedding similarity; we use sparse matrix notation in the pseudocode just for simplicity.

3.4 Complexity Analysis

Here we analyze the computational complexity of each step of REGAL. To simplify notation, we assume both graphs have \( n_1 = n_2 = n \) nodes.

1. Extracting node identity: It takes approximately \( O(n' K d_{avg}^2) \) time, finding neighborhoods up to hop distance \( K \) by joining the neighborhoods of neighbors at the previous hop: formally, we can construct \( R_u^k = \bigcup_{v \in R_u^{k-1}} R_v^k \setminus \bigcup_{v \in R_u^{k-2}} R_v^k \). We could also use breadth-first search from each node to compute the \( k \)-hop neighborhoods in \( O(n'^3) \) worst case time—in practice significantly lower for sparse graphs and/or small \( K \)—but we find that this construction is faster in practice.

2. Computing similarities: We compute the similarities of the length-\( b \) features (weighted counts of node degrees in the \( k \)-hop neighborhoods, split into \( b \) buckets) between each node and \( p \) landmark nodes: this takes \( O(n'p'b) \) time.

3. Obtaining representations: We first compute the pseudoinverse and SVD of the \( p \times p \) matrix \( W \) in time \( O(p^3) \), and then left multiply it by \( C \) in time \( O(n'p'p^2) \). Since \( p' < n' \), the total time complexity for this step is \( O(n'p'^2) \).

4. Aligning embeddings: We construct a \( k \)-d tree and use it to find the top alignment(s) in \( G_2 \) for each of the \( n' \) nodes in \( G_1 \) in average-case time complexity \( O(n' \log n') \). The total complexity is \( O(n' \max(pb, p'^2, K d_{avg}^2 \log n')) \). As we show experimentally, it suffices to choose small \( K \) as well as \( p \) and \( b \) logarithmic in \( n' \). With \( d_{avg} \) often being small in practice, this can yield \( sub-quadratic \) time complexity. It is straightforward to show that the space requirements are sub-quadratic as well.

4 EXPERIMENTS

We answer three important questions about our methods:

(Q1) How does REGAL compare to baseline methods for network alignment on noisy real world datasets (Table 5), with and without attribute information, in terms of accuracy and runtime?

(Q2) How scalable is REGAL?

(Q3) How sensitive are REGAL and xNetMF to hyperparameters?

Experimental Setup. Following the network alignment literature [18, 42], for each real network dataset with adjacency matrix \( A \), we generate a new network with adjacency matrix \( A' = PAP^\top \), where \( P \) is a randomly generated permutation matrix with the nonzero entries representing ground-truth alignments. We add structural noise to \( A' \) by removing edges with probability \( p_a \) without disconnecting any nodes.

For experiments with attributes, we generate synthetic attributes for each node if the graph does not have any. We add noise to these by flipping binary values or choosing categorical attribute values uniformly at random from the remaining possible values with probability \( p_a \). For each dataset and noise level, noise is randomly and independently added.

All experiments are performed on an Intel(R) Xeon(R) CPU E5-1650 at 3.50GHz with 256GB RAM, with hyperparameters \( \delta = 0.01, K = 2, \gamma_r = \gamma_n = 1 \), and \( p = \lceil 10 \log_2 n \rceil \) unless otherwise stated. Landmarks for REGAL are chosen arbitrarily from among the nodes in our graphs, in keeping with the effectiveness and popularity of sampling uniformly at random [6]. In Sec. 4.3, we explore the parameter choices and find that these settings yield stable results at reasonable computational cost.

Baselines. We compare against six baselines. Four are well known existing network alignment methods and two are variants of our proposed framework that match embeddings produced by existing node embedding methods (i.e., not xNetMF). The four existing network alignment methods are: (1) FINAL, which introduces a family of algorithms optimizing quadratic objective functions [42]; (2) NetAlign, which formulates alignment as an integer quadratic programming problem and solves it with message passing algorithms [2]; (3) IsoRank, which solves a version of the integer quadratic program with relaxed constraints [34]; and (4) Klau’s algorithm (Klau), which imposes a linear programming relaxation, decomposes the symmetric constraints and solves it iteratively [16]. These methods all require as input a matrix containing pair alignment information, which we construct from degree similarity, taking the top \( \lceil \log_2 n \rceil \) entries for each node; REGAL, by contrast, does not require prior alignment information.

For the two variants of our framework, which we refer to as (5) REGAL-node2vec and (6) REGAL-struc2vec, we replace our own xNetMF embedding step (i.e., Steps 1 and 2 in REGAL) with existing node representation learning methods node2vec [9] or struc2vec [31]; two recent, state-of-the-art node embedding methods that make a claim about being able to capture some form of structural equivalence. To apply these embedding methods, which were formulated for a single network, we create a single input graph \( G \) by combining the graphs with respective adjacency matrices \( A \) and \( A' \) into one block-diagonal adjacency matrix \( [A \ 0 \ A'] \). Beyond the input, we use their default parameters: 10 random walks
of length 80 for each node to sample context with a window size of 10. For node2vec, we set \( p = q = 1 \) (other values make little difference). For struc2vec, we use the recommended optimizations [31] to compress the degree sequences and reduce the number of node comparisons, which were found to speed up computation with little effect on performance [31]. As we do for our xNetMF method, we consider a maximum hop distance of \( K = 2 \).

**Metrics.** We compare REGAL to baselines with two metrics: alignment accuracy, which we take as \((\text{# correct alignments}) / (\text{total # alignments})\), and runtime. When computing results, we average over 5 independent trials on each dataset at each setting (with different random permutations and noise additions) and report the mean result and the standard deviation (as bars around each point in our plots.) We also show where REGAL’s soft alignments contain the “correct” similarities within its top \( \alpha < n \) choices using the more general top-\( \alpha \) accuracy: \((\text{# correct alignments in top-\( \alpha \) choices}) / (\text{total # alignments})\). This metric does not apply to the existing network alignment baselines that do not directly match node embeddings and only find hard alignments.

### 4.1 Q1: Comparative Alignment Performance

To assess the comparative performance of REGAL versus existing network alignment methods on a variety of challenging datasets, we perform two experiments studying the effects of structural and attribute noise, respectively.

#### 4.1.1 Effects of structural noise

In this experiment we study how well REGAL matches nodes based on structural identity alone. This also allows us to compare to the baseline network alignment methods NetAlign, IsoRank, and Klau, as well as the node embedding methods node2vec and struc2vec, none of which was formulated to handle or align attributed graphs (which we study in Sec. 4.1.2). As we discuss further below, REGAL is one of the fastest network alignment methods, especially on large datasets, and has comparable or better accuracy than all baselines.

#### 4.1.2 Effects of attribute noise

In this experiment we study how network alignment methods on a variety of challenging datasets, and how they perform two experiments studying the effects of structural and attribute noise, respectively.

#### 4.1.3 Performance

We compare REGAL to baselines with two metrics: alignment accuracy and runtime. When computing results, we average over 5 independent trials on each dataset at each setting (with different random permutations and noise additions) and report the mean result and the standard deviation (as bars around each point in our plots.) We also show where REGAL’s soft alignments contain the “correct” similarities within its top \( \alpha < n \) choices using the more general top-\( \alpha \) accuracy: \((\text{# correct alignments in top-\( \alpha \) choices}) / (\text{total # alignments})\). This metric does not apply to the existing network alignment baselines that do not directly match node embeddings and only find hard alignments.

#### 4.1.4 Runtime

In Table 4, we compare the average runtimes of all different methods across noise levels. We observe that REGAL scales significantly better using xNetMF than when using other node embedding methods. Notably, REGAL is \( 6-8 \times \) faster than REGAL-node2vec and 22-31\( \times \) faster than REGAL-struc2vec. This is expected as both dynamic time warping (in struc2vec) and context sampling for SGNS (in struc2vec and node2vec) come with large computational costs. REGAL, at the cost of some robustness to high levels of noise, avoids both the variance and computational expense of random-walk-based sampling. This is a significant benefit that allows REGAL to achieve up to an order of magnitude speedup over other methods node2vec and struc2vec, none of which was formulated to handle or align attributed graphs (which we study in Sec. 4.1.2).

#### 4.1.5 Results

**Results. (1) Accuracy.** The accuracy results on several datasets are shown in Figure 4. The structural embedding REGAL variants consistently perform best. Both REGAL (matching our proposed xNetMF embeddings) and REGAL-struc2vec are significantly more accurate than all non-representation learning baselines across noise levels and datasets. As expected, REGAL-node2vec does hardly better than random chance because rather than preserving structural similarity, it preserves similarity to nodes based on their proximity to each other, which means there is no way of identifying similarity to corresponding nodes in other, disconnected graphs (even when we combine them into one large graph, because they form disconnected components.) This major limitation of embedding methods that use proximity-based node similarity criteria [11] justifies the need for structural embeddings for cross-network analysis.

Between REGAL and REGAL-struc2vec, the two highest performers, REGAL performs better with lower amounts of noise. This is likely because struc2vec’s randomized context sampling introduces some variance into the representations that xNetMF does not have, as nodes that should match will have different embeddings not only because of noise, but also because they had different contexts sampled. With higher amounts of noise (4-5%), REGAL outperforms REGAL-struc2vec in speed, but at the cost of some accuracy. It is also worth noting that their accuracy margin is smaller for larger graphs. On larger datasets, our simple and fast logarithmic binning scheme (Step 1 in Sec. 3.1) provides a robust enough way of comparing nodes with high expected degrees. However, on small graphs with a few thousand nodes and edges, it appears that struc2vec’s use of dynamic time warping (DTW) better handles misalignment of degree sequences from noise because it is a nonlinear alignment scheme. Still, we will see that REGAL is significantly faster than its struc2vec variant, since DTW is computationally expensive [31], as is context sampling and SGNS training.

**Results. (2) Runtime.** In Table 4, we compare the average runtimes of all different methods across noise levels. We observe that REGAL scales significantly better using xNetMF than when using other node embedding methods. Notably, REGAL is \( 6-8 \times \) faster than REGAL-node2vec and 22-31\( \times \) faster than REGAL-struc2vec. This is expected as both dynamic time warping (in struc2vec) and context sampling for SGNS (in struc2vec and node2vec) come with large computational costs. REGAL, at the cost of some robustness to high levels of noise, avoids both the variance and computational expense of random-walk-based sampling. This is a significant benefit that allows REGAL to achieve up to an order of magnitude speedup over...
the other node embedding methods. Additionally, REGAL is able to leverage the power of node representations and also use attributes, unlike the other representation learning methods.

Comparing to baselines that do not use representation learning, we see that REGAL is competitive in terms of runtime as well as significantly more accurate. REGAL is consistently faster than FINAL and Klau, the next two best-performing methods by accuracy (NetAlign is virtually tied for third place with Klau on all datasets). Although NetAlign runs faster than REGAL on small datasets like Arenas, on larger datasets like Arxiv NetAlign’s message passing becomes expensive. Finally, while IsoRank is consistently the fastest method, it performs among the worst on all datasets in accuracy. Thus, we can see that our REGAL framework is also one of the fastest network alignment methods as well as the most accurate.

4.1.2 Effects of attribute-based noise. In the second experiment, we study REGAL’s comparative sensitivity to \( p_a \) when we use node attributes. Here we compare REGAL to FINAL because it is the only baseline that handles attributes. We also omit embedding methods other than xNetMF, since they operate on plain graphs.

We study a subnetwork of a larger DBLP collaboration network extracted in [42] (Table 5). This dataset has 1 node attribute with 29 values, corresponding to the top conference in which each author (a node in the network) published. This single attribute is quite discriminatory: when so many possible attribute values, a comparatively smaller number of nodes share the same value. We add \( p_a = 0.01 \) structural noise to randomly generated permutations.

We also increase attribute information by increasing the number of attributes. To do so, we simulate different numbers of binary attributes. We study somewhat higher levels of attribute noise, as they are not strictly required for network alignment.

Results. In Figure 5, we see that REGAL mostly outperforms FINAL in the presence of attribute noise (both for real and multiple synthetic attributes), or in the case of limited attribute information (e.g., only 1-3 binary attributes in Fig. 5a-5c). This is because FINAL relies heavily on attributes, whereas REGAL uses structural and attribute information in a more balanced fashion.

While FINAL achieves slightly higher accuracy than REGAL with abundant attribute information from many attributes or attribute values and minimal noise (e.g. the real attribute with 29 values in Figure 5d, or 5 binary attributes in Figure 5c), this is expected due to FINAL’s reliance on attributes. Also, in Figure 5e where we plot the runtime with respect to number of attributes : attribute values, we see FINAL incurs significant runtime increases as it uses extra attribute information. Even without these added attributes, REGAL is up to two orders of magnitude faster than FINAL.

4.2 Q2: Scalability

To analyze the scalability of REGAL, we generate Erdős-Rényi graphs with \( n = 100 \) to 1,000,000 nodes and constant average degree 10, along with one binary attribute. We generate a randomized, noisy permutation \( (p_x = 0.01, p_a = 0.05) \) and look for the top \( \alpha = 1 \) alignments. Thus, we embed both graphs—double the number of nodes in a single graph. Figure 7 shows the runtimes for the major steps of our methods.

Results. We see that the total runtimes of REGAL’s steps are clearly sub-quadratic, which is rare for alignment tasks. In practice this means that REGAL can scale to very large networks. The dominant step is computing \( O(n \log n) \) similarities to landmarks in C and using this to form the Nyström-based representation. The alignment time complexity grows the most steeply, as the dimensionality \( p \) grows with the network size and increasingly affects lookup times. In practice, though, the alignment adds little overhead time, even for the largest graph, because of the \( k-d \) tree. Without it, REGAL runs out of memory on 100K or more nodes.

From a practical perspective, while our current implementation is single-threaded, many steps—including the expensive embedding construction and alignment steps—are easily and trivially parallelizable, offering possibilities for even greater speedups.

4.3 Q3: Sensitivity Analysis

To understand how REGAL’s hyperparameters affect performance, we analyze accuracy by varying hyperparameters in several experiments. For brevity, we report results at \( p_a = 0.01 \) and with a single binary noiseless attribute, although further experiments with different settings yielded similar results. Overall we find that REGAL is robust to different settings and datasets, indicating that REGAL can be applied readily to different graphs without requiring excessive domain knowledge or fine-tuning.

Results. (1) Discount factor \( \delta \) and max hop distance \( K \). Figures 6a and 6b respectively show the performance of REGAL as a function of \( \delta \), the discount factor on further hop distances, and \( K \), the maximum hop distance to consider. We find that some higher-order
structural information does help (thus K = 2 performs slightly better than K = 1), but only up to a point. Beyond approximately 2 layers out, the structural similarity is so tenuous that itprimarily adds noise to the neighborhood degree distribution (furthermore, computing further hop distances adds computational expense). Choosing δ between 0.01–0.1 tends to yield best performance. Larger discount factors δ tend to do poorly, though extremely small values may lose higher-order structural information.

(2) Weights of structural γs and attributed γa similarity. Next, we explore how to set the coefficients on the terms in the similarity function weighting structural and attribute similarity, which also governs a tradeoff between structural and attribute identity. In Figs. 6c and 6d we respectively vary γs and γa while setting the other to be 1. In general, setting these parameters to be 1, our recommended default value, does fairly well. Significantly larger values yield less stable performance.

(3) Dimensionality of embeddings p. To study the effects of the rank of the implicit low-rank approximation, which is also the dimensionality of the embeddings, we set the number of landmarks p equal to ⌊t log2 n⌋ and vary t. Figure 8a shows that the accuracy is generally highest for the highest values of t, but Figure 8b shows the expected increase in REGAL’s runtime as more similarities are computed in C and higher-dimensional embeddings are compared. To spare no expense in maximizing accuracy we use t = 10. However, fewer landmarks still yield almost as high accuracy if computational constraints or high dimensionality are issues.

(4) Top-α accuracy. It is worth studying not just the proportion of correct hard alignments, but also the top-α scores of the soft alignments that REGAL can return. We perform alignment without attributes on a large Facebook subnetwork [37] and visualize the top-1, top-5, and top-10 scores in Fig. 6e. Across noise settings, the top-α scores are considerably several percentage points higher than the top-1 scores, indicating that even when REGAL misaligns a node, it often still recognizes the similarity of its true counterpart. REGAL’s ability to find soft alignments could be valuable in many applications, like entity resolution across social networks [18].

5 CONCLUSION

Motivated by the numerous applications of network alignment in social, natural, and other sciences, we proposed REGAL, a network alignment framework that leverages the power of node representation learning by aligning nodes via their learned embeddings. To efficiently learn node embeddings that are comparable across multiple networks, we introduced xNetMF within REGAL. To the best of our knowledge, we are the first to propose an unsupervised representation learning-based network alignment method.

Our embedding formulation captures node similarities using structural and attribute identity, making it suitable for cross-network analysis. Unlike other embedding methods that sample node context with computationally expensive and variance-inducing random walks, our extension of the Nyström low-rank approximation allows us to implicitly factorize a similarity matrix without having to fully construct it. Furthermore, we showed that our formulation is a matrix factorization perspective on the skip-gram objective optimized over node context sampled from a similarity graph. Experimental results showed that REGAL is up to 30% more accurate than baselines and 30× faster in the representation learning stage. Future directions include extending our techniques to weighted networks and incorporating edge signs or other attributes.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. IIS 1743088, an Adobe Digital Experience research faculty award, and the University of Michigan. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding parties. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

A CONNECTIONS: xNetMF and SGNS

Here we unpack the key components of the struc2vec framework [31], a random walk-based structural representation learning approach, and we find a matrix factorization interpretation at the heart of it. Given a (single-layer) similarity graph $S$, for each node $v$, struc2vec samples context nodes $C$ with random walks of length $l$ starting from $v$. The probability of going from node $u$ to node $v$ is proportional to the nodes’ (structural) similarity $\gamma_{uv}$. This yields a co-occurrence matrix $D = \gamma_{uv}$ is the number of times node $v$ was visited in context of node $u$. Afterward, struc2vec optimizes a skip-gram objective function with negative sampling (SGNS):

$$\max_{\theta, \gamma} \sum_{v \in V, c \in C} \#(y, c) \log \sigma(y^T \gamma) + \ell \cdot \mathbb{E}_{u \sim P_D} \log \sigma(-y^T \gamma') \tag{3}$$

where $y$ and $c$ are the embeddings of a node $y$, and its context node $c$, resp.; $P_D(c) = \sum_{y \in V} \#(y, c)/\sum_{y \in V, c \in C} \#(y, c)$ is the empirical probability that a node is sampled as some other node’s context; and $\sigma(x) = (1 + e^{-x})^{-1}$ is the sigmoid function. Analysis of SGNS for word embeddings [25] showed under some assumptions on the upper bound of the co-occurrence count between two words that the objective of SGNS in Eq. (3) is equivalent to matrix factorization of the co-occurrence matrix $D$, or $MF(D, Y^C)$. Here MF is the objective of matrix factorization on $D$ (formally defined in [25], but in practice other matrix factorization techniques work well).

Now, under these assumptions, we show a connection between optimizing Eq. (3) with context sampled from the similarity graph (as in struc2vec), and factorizing the graph (as in xNetMF).

**Lemma A.1.** Equation (3), defined over a context sampled by performing $m$ length-1 random walks per node over $S$, is equivalent to MFS($Y^C$) in the limit as $m$ goes to $\infty$, up to scaling of $S$.

Proof. This follows from the Law of Large Numbers. As $m \to \infty$, the co-occurrence matrix $D$ converges to its expectation. This is just $m \cdot S$, since $d_{ij}$ is the # of times node $v_j$ is sampled in a random walk of length 1 from $v_i$, which is equal to the # of walks from node $v_i$ times the probability that the walk goes to $v_j$ from $v_i$, or $m \cdot s_{ij}$. (Since MF is invariant to scaling, we normalize $D$ w.r.t. $m$.)

Note that in struc2vec, increasing $m$ to sample more context reduces variance in $D$, but increasing $\ell$ simply causes the random walks to move further from the original node $u$ and sample context based on similarity to more structurally distant nodes. Lemma A.1 connects xNetMF to a version of struc2vec with maximal $m$ and minimal $\ell$, further justifying its success by comparison.