Weisfeiler-Lehman Neural Machine for Link Prediction

Muhan Zhang
Department of Computer Science and Engineering
Washington University in St. Louis
muhan@wustl.edu

Yixin Chen
Department of Computer Science and Engineering
Washington University in St. Louis
chen@cse.wustl.edu

ABSTRACT

In this paper, we propose a next-generation link prediction method, Weisfeiler-Lehman Neural Machine (WLNM), which learns topological features in the form of graph patterns that promote the formation of links. WLNM has unmatched advantages including higher performance than state-of-the-art methods and universal applicability over various kinds of networks. WLNM extracts an enclosing subgraph of each target link and encodes the subgraph as an adjacency matrix. The key novelty of the encoding comes from a fast hashing-based Weisfeiler-Lehman (WL) algorithm that labels the vertices according to their structural roles in the subgraph while preserving the subgraph's intrinsic directionality. After that, a neural network is trained on these adjacency matrices to learn a predictive model. Compared with traditional link prediction methods, WLNM does not assume a particular link formation mechanism (such as common neighbors), but learns this mechanism from the graph itself. We conduct comprehensive experiments to show that WLNM not only outperforms a great number of state-of-the-art link prediction methods, but also consistently performs well across networks with different characteristics.

CCS CONCEPTS

• **Information systems → Data mining**; • **Computing methodologies → Supervised learning**;

KEYWORDS

link prediction; graph labeling; color refinement; neural network

ACM Reference format:
Muhan Zhang and Yixin Chen. 2017. Weisfeiler-Lehman Neural Machine for Link Prediction. In *Proceedings of KDD '17, Halifax, NS, Canada, August 13-17, 2017,* 9 pages.
https://doi.org/10.1145/3097983.3097996

1 INTRODUCTION

Link prediction [1] is attracting increasing interests among data mining and machine learning communities. It has many applications, such as friend recommendation in social networks [2], product recommendation in e-commerce [3], knowledge graph completion [4], finding interactions between proteins [5], and recovering missing reactions in metabolic networks [6]. While many sophisticated models such as stochastic block models [5] and probabilistic matrix factorization [7] have been developed, some simple heuristics such as common neighbors and Katz index work surprisingly well in practice and are often more interpretable and scalable. For instance, the common neighbor heuristic assumes that two nodes are more likely to have a link if they have more common neighbors. Although simple, this heuristic has been shown to perform very well on social networks [1]. Other successful heuristics include AA [2], RA [8], Katz [9] as well as many carefully calculated node proximity scores based on network topology or random walks.

However, a significant limitation of these heuristics is that they lack universal applicability to different kinds of networks. For example, common neighbors may work well when predicting friendships in social networks or predicting coauthorships in collaboration networks, but has been shown to have poor performance on electrical grids and biological networks [10]. On the other hand, Average Commute Time, a.k.a. resistance distance [11], has exceptional performance on predicting power grids and router-level Internets, but has poor results on social networks. A survey paper compared over 20 different heuristics and found that none of them performs consistently well across all networks [10]. This implies the need to manually choose different heuristics for different networks based on prior beliefs or expensive trial and error.

Can we automatically learn suitable heuristics from a network itself? The answer is yes, since these heuristics are after all extracted from the network topology. By extracting local patterns for each link, we should be able to learn which patterns foster the formation of a link. This way, various heuristics embedded in the local patterns can be learned automatically, avoiding the need to manually select heuristics. Moreover, for those networks on which no existing heuristic works well, we can learn new heuristics that suit them. Our goal in this paper is to design such a universal model.

We propose a new link prediction method called Weisfeiler-Lehman Neural Machine (WLNM). For each target link, WLNM first extracts a subgraph in its neighborhood, which we call the *enclosing subgraph* of a link. WLNM then represents the enclosing subgraph as an adjacency matrix. After that, a neural network is trained on these adjacency matrices to learn a link prediction model. Figure 1 illustrates the proposed framework.

To encode each enclosing subgraph, the key issue is to decide the ordering of graph vertices. The goal of *graph labeling* is to assign nodes of two different enclosing subgraphs to similar indices in respective adjacency matrices if and only if their structural roles within the graphs are similar. Since machine learning models read data sequentially, a stable ordering based on structural roles of vertices is crucial for learning meaningful models.

The Weisfeiler-Lehman (**WL**) algorithm [12] is a graph labeling method which determines vertex ordering based on graph topology. The classical WL algorithm works as follows. Initially, all vertices get the same label. Then, vertices iteratively concatenate their own

KDD '17, August 13-17, 2017, Halifax, NS, Canada

ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3097996

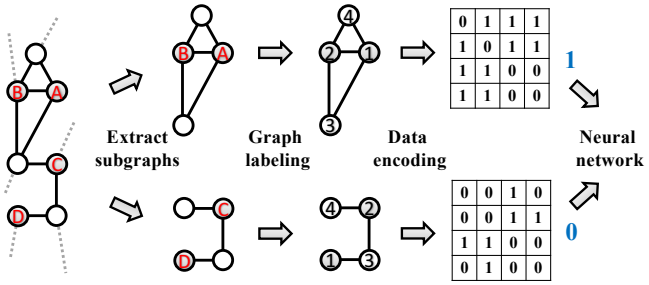


Figure 1: An illustration of WLNM. Given a network, WLNM first samples a set of positive links (illustrated as link (A, B)) and negative links (illustrated as link (C, D)) as training links, and extracts an enclosing subgraph for each link. Graph labeling is used to decide the vertex ordering and adjacency matrix. The resulting (matrix, label) pairs are fed into a neural network for link prediction.

labels and their direct neighbors' labels as their signature strings and compress these signature strings into new, short labels until convergence. In the end, vertices with identical structural roles will get the same labels.

A limitation of the classical WL algorithm is that it requires the storing and sorting of possibly long signature strings in each iteration, which is time consuming. On the other hand, a hashing-based WL [13] is much faster, but is no longer stable: the vertex ordering is not preserved between iterations. To address the above challenge, we propose a novel PALETTE-WL graph labeling algorithm, which combines the efficiency of the hashing-based WL and the order-preserving property of the classical WL. PALETTE-WL first colors subgraph vertices according to their distance to the target link, and then iteratively refines the initial colors so that the relative color ordering is preserved. Our results prove that the PALETTE-WL algorithm leads to very good enclosing subgraph representations and is computationally efficient.

To learn nonlinear topological features from the enclosing subgraphs, neural network is used for its exceptional expressing power. Our experiments show that neural networks achieve superior link prediction performance than heuristic methods, especially on some datasets where all existing methods perform poorly.

WLNM has a few distinctive advantages. 1) **Higher performance**—WLNM uses neural networks to learn sophisticated topological features which simple heuristics cannot express. It outperforms all baseline methods in almost all datasets we have tested; 2) **Universality**—WLNM automatically learns topological features, avoiding the need to choose heuristics or do feature selection/engineering for different networks. Empirical results confirm that WLNM consistently performs well across various networks, while most other methods perform well only on a few networks and poorly on others.

We summarize our contributions as follows. 1) We propose Weisfeiler-Lehman Neural Machine (WLNM), a novel link prediction framework to automatically learn topological features from networks. 2) We propose a novel graph labeling method, PALETTE-WL, to efficiently encode enclosing subgraphs into adjacency matrices so that neural networks can learn meaningful patterns. 3) We conduct extensive experiments on various kinds of real-world networks and compare WLNM to 12 heuristic and latent feature methods. WLNM outperforms state-of-the-art link prediction methods and offers great universality across various networks.

Table 1: Popular Heuristics for Link Prediction

Name	Formula	Order
common neighbors	$\lvert\Gamma(x) \cap \Gamma(y)\rvert$	first
Jaccard	$\frac{\lvert\Gamma(x)\cap\Gamma(y)\rvert}{\lvert\Gamma(x)\cup\Gamma(y)\rvert}$	first
preferential attachment	$\lvert\Gamma(x)\rvert \cdot \lvert\Gamma(y)\rvert$	first
Adamic-Adar	$\sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{\log\lvert\Gamma(z)\rvert}$	second
resource allocation	$\sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{\lvert\Gamma(z)\rvert}$	second
Katz	$\sum_{l=1}^{\infty} \beta^l \lvert\text{path}(x, y) = l\rvert$	high
PageRank	$q_{xy} + q_{yx}$	high
SimRank	$\gamma \frac{\sum_{a\in\Gamma(x)}\sum_{b\in\Gamma(y)}\text{score}(a,b)}{\lvert\Gamma(x)\rvert\cdot\lvert\Gamma(y)\rvert}$	high
resistance distance	$\frac{1}{l^+_{xx}+l^+_{yy}-2l^+_{xy}}$	high

Notes: $\Gamma(x)$ denotes the neighbor set of vertex x. $\beta < 1$ is a damping factor. $\lvert\text{path}(x, y) = l\rvert$ counts the number of length-l paths between x and y. q_{xy} is the stationary distribution probability of y under the random walk from x with restart, see [14]. SimRank score is a recursive definition. l^+_{xy} is the (x, y) entry of the pseudoinverse of the graph's Laplacian matrix.

2 PRELIMINARIES

In this section, we introduce some background knowledge in link prediction and graph theory, which are important for understanding the proposed WLNM method.

2.1 Heuristic methods for link prediction

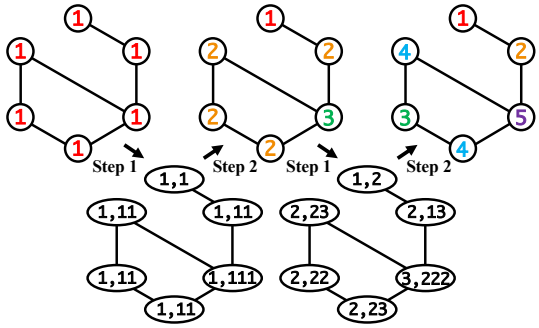
A large category of link prediction methods are based on some heuristics that measure the proximity between nodes to predict whether they are likely to have a link. Popular heuristics include: common neighbors (CN), Adamic-Adar (AA) [2], preferential attachment (PA) [15], resource allocation (RA) [8], Katz [9], PageRank [14], SimRank [16], resistance distance [11], and their numerous variants. Liben-Nowell and Kleinberg [1] first studied their link prediction performance on social networks. Empirical comparisons of these heuristics on different networks can be found in [10, 17]. We group link prediction heuristics into three classes: first-order, second-order and high-order methods, based on the most distant node necessary for computing the heuristic. For example, common neighbor is a first-order heuristic, since it only involves the direct neighbors of the two nodes. Katz index is a high-order heuristic, because one needs to search the entire graph for all possible paths between two vertices. Table 1 summarizes nine popular heuristics, which will be used as baselines in our experiments.

2.2 Graphs

A network can be represented as a graph $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ is the set of vertices and $E \subseteq V \times V$ is the set of links. A graph can be represented by an adjacency matrix A, where $A_{i,j} = 1$ if there is a link from i to j and $A_{i,j} = 0$ otherwise. We say i and j are adjacent if $A_{i,j} = 1$. If the links are undirected, A will be symmetric. In this paper, we consider undirected networks, although our model can be easily generalized to directed networks. We use $\Gamma(x)$ or $\Gamma^1(x)$ to denote the set of 1-hop neighbors of a vertex $x \in V$. We use

Algorithm 1 Weisfeiler-lehman Graph Labeling

input: graph $G = (V, E)$, initial colors $c^0(v) = 1$ for all $v \in V$
output: final colors $c(v)$ for all $v \in V$
let $c(v) = c^0(v)$ for all $v \in V$
while $c(v)$ has not converged **do**
  for each $v \in V$ **do**
    collect a multiset $\{c(v')|v' \in \Gamma(v)\}$ containing its neighbors' colors
    sort the multiset in ascending order
    concatenate the sorted multiset to $c(v)$ to generate a signature string $s(v) = \langle c(v), \{c(v')|v' \in \Gamma(v)\}_{\text{sort}}\rangle$
  end for
  sort all $s(v)$ in lexicographical ascending order
  map all $s(v)$ to new colors 1,2,3,... sequentially; same strings get the same color
end while



Step 1: generate signature strings. Step 2: sort signature strings and recolor.

Figure 2: Illustration of two iterations of the WL algorithm for a graph. The vertices in the upper left graph are initially all colored 1. In each iteration, step 1 calculates the WL signature string for each vertex by concatenating the color of the vertex and the (sorted) colors of the vertex's neighbors. Step 2 recolors the graph according to these WL signatures.

$\Gamma^d(x)$ to denote the set of vertices whose distance to x is less than or equal to d, $d = 1, 2, 3, \cdots$.

2.3 The Weisfeiler-Lehman algorithm

A **graph labeling** function is a map $l : V \to C$ from vertices V to an ordered set C, conventionally called **colors** in literature. In this paper, we adopt the set of integer colors starting from 1. If l is injective, then C can be used to uniquely determine the vertex order in an adjacency matrix.

Our proposed Palette-WL graph labeling method is based on the 1-dimensional Weisfeiler-Lehman (WL) algorithm [12], shown in Algorithm 1. Widely used in graph isomorphism checking, WL belongs to a class of **color refinement** algorithms that iteratively update vertex colors until a fixed point is reached.

The main idea of WL is to iteratively augment vertex labels using their neighbors' labels and compress the augmented labels into new labels until convergence. At first, all vertices are set to the same color 1. For each vertex, it gets a *signature string* by concatenating its own color and the sorted colors of its immediate neighbors. Vertices are then sorted by the ascending order of their signature strings and assigned new colors $1, 2, 3, \cdots$. Vertices with the same signature strings get the same color.

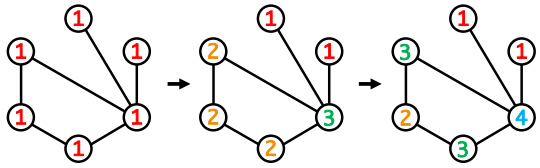


Figure 3: Illustration of the WL coloring for another graph. Comparing the vertex orderings in Figure 2 and Figure 3, we see that vertices with similar structural roles have similar relative rankings.

For example, assume vertex x has color 2 and its neighbors have colors $\{3, 1, 2\}$ respectively, and vertex y has color 2 and its neighboring colors are $\{2, 1, 2\}$. The signature strings for x and y are $\langle 2, 123\rangle$ and $\langle 2, 122\rangle$, respectively. Since $\langle 2, 122\rangle$ is smaller than $\langle 2, 123\rangle$ lexicographically, y will be assigned a smaller color than x in the next iteration. Such process is iterated until vertex colors stop changing. Figure 2 shows an example. All vertices are initially colored 1 and finally colored by a richer set $\{1, \cdots, 5\}$.

One key benefit of the WL algorithm is that the final colors encode the structural roles of vertices inside a graph and define a relative ordering for vertices (with ties)—vertices with the same final color share the same structural role within a graph. Moreover, this relative ordering for vertices is consistent across different graphs—e.g., if vertex v in G and vertex v' in G' share similar structural roles in their corresponding graphs, they will have similar relative positions in their respective orderings, which is shown in Figure 3. The structure-encoding property of WL is essential for its success in graph kernel design [18], which measures graph similarity by counting vertices' matching WL colors. Recently, WL is also used in graph CNNs to define a global ordering for sequentially moving convolutional filters along vertices and a local ordering for reading each receptive field [19].

3 WEISFEILER-LEHMAN NEURAL MACHINE (WLNM)

In this section, we propose our Wlnm model. Wlnm is a neural network model combined with encoded subgraph patterns. It automatically learns topological features that promote the formation of links from each link's local subgraph pattern.

To encode the subgraph patterns, we propose Palette-WL, a variant of WL that is fast and order-preserving. Palette-WL leverages the ability of the classical WL to label vertices according to their structural roles, but also preserves vertices' initial relative ordering defined by their distance to the target link, a property that is crucial for link prediction. Wlnm further leverages the superior expressive power of neural networks to learn possibly complicated link formation mechanisms which are difficult to model by heuristic scores. As a result, Wlnm has remarkable prediction performance and universality.

Wlnm includes the following three main steps:

(1) Enclosing subgraph extraction, which generates K-vertex neighboring subgraphs of links.

Algorithm 2 Enclosing Subgraph Extraction

input: target link (x, y), network G, integer K
output: enclosing subgraph $G(V_K)$ for (x, y)
$V_K = \{x, y\}$
$fringe = \{x, y\}$
while $|V_K| < K$ and $|fringe| > 0$ **do**
  $fringe = (\bigcup_{v \in fringe} \Gamma(v)) \setminus V_K$
  $V_K = V_K \cup fringe$
end while
return enclosing subgraph $G(V_K)$

(2) Subgraph pattern encoding, which represents each subgraph as an adjacency matrix whose vertex ordering is given by our Palette-WL graph labeling algorithm.
(3) Neural network training, which learns nonlinear graph topological features for link prediction.

3.1 Enclosing subgraph extraction

To learn topological features from a network, Wlnm extracts an enclosing subgraph for each link, and use these ⟨subgraph, link⟩ pairs as training data. The enclosing subgraph of each link describes the "surrounding environment" of that link, which we assume contains topological information deciding whether a link is likely to exist.

For a given link, its **enclosing subgraph** is a subgraph within the neighborhood of that link. The size of the neighborhood is described by the number of vertices in the subgraph, which is denoted by a user-defined integer K. We describe the procedure for extracting the enclosing subgraph in the following.

For a given link between x and y, we first add their 1-hop neighbor vertices $\Gamma(x)$ and $\Gamma(y)$ to an ordered node list V_K. Then, vertices in $\Gamma^2(x), \Gamma^2(y), \Gamma^3(x), \Gamma^3(y), \cdots$, are iteratively added to V_K until $|V_K| \geq K$ or there are no more neighbors to add. Algorithm 2 shows the enclosing subgraph extraction process.

After running the extraction algorithm, the number of vertices in V_K may not be exactly K. One way to unify the size is to discard the last added $|V_K| - K$ vertices if $|V_K| > K$. In this paper, we adopt a different strategy. Inspired by [19], we first use graph labeling to impose an ordering for V_K, and then reorder V_K using this order. After that, if $|V_K| > K$, the bottom $|V_K| - K$ vertices are discarded. If $|V_K| < K$, we add $K - |V_K|$ dummy nodes to V_K. This way, the sizes of different enclosing subgraphs are unified to K.

When $K \geq |\Gamma(x) \cup \Gamma(y) \cup x \cup y|$, the enclosing subgraph contains all the information needed for calculating first-order heuristics in Table 1. When $K \geq |\Gamma^2(x) \cup \Gamma^2(y) \cup x \cup y|$, it contains the information needed for second-order heuristics. When K equals $|V|$, the extracted subgraph encompasses all first-order, second-order and high-order heuristics. This gives an intuitive explanation why Wlnm outperforms heuristic methods.

3.2 Subgraph pattern encoding

Subgraph pattern encoding is to represent each enclosing subgraph as an adjacency matrix with a particular vertex ordering, so that the neural network in Wlnm can read the data in sequence. We illustrate the process of subgraph pattern encoding in Figure 4, and explain the details below.

3.2.1 Palette-WL for vertex ordering. We use graph labeling to determine the vertex ordering for each enclosing subgraph. To facilitate the training of neural networks, the vertex orderings generated by the graph labeling algorithm should be consistent across different subgraphs, e.g., vertices receive similar rankings if their relative positions and structural roles within their respective subgraphs are also similar. We will describe our proposed Palette-WL algorithm and explain why we adopt it in the following.

We first formally state our two intuitive requirements for the graph labeling algorithm used here as follows.

(1) It should impose vertex orderings such that two nodes with similar structural roles in their respective enclosing subgraphs have similar rankings.
(2) It should distinguish the "target link" in each enclosing subgraph and preserve the topological directionality within the enclosing subgraph.

The first requirement is important, since it allows machine learning models to sequentially read vertices of enclosing subgraphs in a stable order. It can be satisfied by the classical WL algorithm, since WL ranks vertices according to their structural roles (as exemplified by Figure 2 and 3). However, the classical WL does not meet the second requirement. Namely, it cannot **distinguish the target link** from other parts of the enclosing subgraph. This is because WL treats all vertices equally in the beginning. From the final WL colors, we cannot tell which colors encode the two nodes of the target link. Such a limitation will make the training meaningless.

We further explain the importance of the second requirement as follows. Unlike ordinary graphs, enclosing subgraphs have intrinsic directionality: at the center is the target link, other vertices and edges are iteratively added outwards based on their distance to the central link. A good graph labeling algorithm should be able to reflect this directionality, e.g., 1) the two central vertices always have the smallest colors; 2) vertices closer to the center link have smaller colors than farther ones. Such directionality is crucial for defining meaningful vertex orderings. If a graph labeling method does not keep this directionality, the generated vertex representation may be very poor for link prediction.

We will propose a Palette-WL algorithm that meets both requirements above. To formalize our analysis, we first give the definition of *color-order preservingness*.

Definition 3.1. An iterative graph labeling algorithm is color-order preserving if: given any two vertices v_a and v_b, if v_a has a smaller color than v_b at an iteration, then v_a gets a smaller color than v_b in the next iteration.

A color-order preserving algorithm has the following benefit when used as a graph labeling method:

Corollary 3.2. *If a graph labeling algorithm is color-order preserving, then the vertices' final color ordering still observes their initial color ordering. In other words, if vertex v_a's initial color is smaller than v_b's, v_a's final color is still smaller than v_b's final color.*

Remember that we need the final vertex ordering of an enclosing subgraph to reflect the vertices' distance to the target link. This can be achieved if we initially label vertices based on the ascending order of their distance to the target link, and then run a color-order preserving algorithm to refine their labels.

For instance, we can initially assign color 1 to the two vertices of the target link, color 2 to the link's 1-hop neighbors, color 3 to

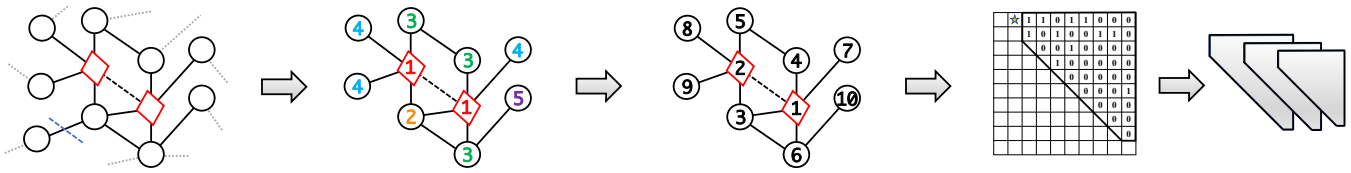


Figure 4: Illustration of the WLNM procedure: enclosing subgraph extraction (the leftmost figure), subgraph pattern encoding (the middle three figures), and neural network training (the rightmost figure).

the link's 2-hop neighbors, etc., and then run color refinement on these initial colors. The color-order preserving property ensures that the final labels still observe the distance ordering. Moreover, since the two vertices from the target link have the smallest initial color, they are guaranteed to have smaller final colors than all other vertices. This means that a target link is always encoded as $A_{1,2}$ in its enclosing subgraph.

Color-order preservingness is a much desired property for the graph labeling algorithm in WLNM to have. Fortunately, we have the following theorem.

THEOREM 3.3. *The classical 1-dimensional WL algorithm is color-order preserving.*

PROOF. At the i^{th} iteration of the WL algorithm (given in Algorithm 1), consider any pair of vertices v_a and v_b, and assume their current colors are $c^i(v_a)$ and $c^i(v_b)$, respectively. Their signature strings $s^i(v_a)$ and $s^i(v_b)$ are $\langle c^i(v_a), \{c^i(x)|x \in \Gamma(v_a)\}_{\text{sort}}\rangle$ and $\langle c^i(v_b), \{c^i(x)|x \in \Gamma(v_b)\}_{\text{sort}}\rangle$, respectively. If $c^i(v_a) < c^i(v_b)$, then $s^i(v_a)$ is lexicographically smaller than $s^i(v_b)$ regardless of their latter letters. Therefore, we have $c^{i+1}(v_a) < c^{i+1}(v_b)$. □

Thus, the classical WL algorithm is an eligible graph labeling algorithm for WLNM. However, it requires storing, reading, and sorting of the vertices' signature strings, which is often prohibitively expensive since the signature strings can be very long for nodes with high degree.

Recently, a fast hashing-based WL algorithm was proposed [13]. It uses a perfect hash function $h(x)$ to map unique signatures to unique real values. As a result, vertices can be iteratively partitioned using their hash values instead of the signature strings, which is shown to be much faster than the classical WL algorithm [13].

The hash function for vertex x is as follows:

$$h(x) = c(x) + \sum_{z \in \Gamma(x)} \log(\mathcal{P}(c(z))), \tag{1}$$

where $c(x)$ and $c(z)$ are integer colors, $\mathcal{P}$ is the list of all primes, where $\mathcal{P}(n)$ is the n^{th} prime number. It can be shown that, given two vertices x and y, $h(x) = h(y)$ if and only if: 1) $c(x) = c(y)$; and 2) $\Gamma(x)$ and $\Gamma(y)$ contain the same colors with the same cardinality (same WL signature ⇔ same new color).

Albeit much faster than the classical WL, the above hashing-based WL is not color-order preserving. In other words, although vertices can be partitioned according to structural roles, their final colors do not define a meaningful ordering. Besides, the colors generated by the above WL sometimes do not converge in our experiments. That is, two vertices may start to exchange their colors from some iteration on and never stop. To address the above issues as well as preserving the efficiency of the hashing-based WL, we propose the PALETTE-WL algorithm. It is a color refinement method equipped with a modified hash function:

$$h(x) = c(x) + \frac{1}{\left\lceil \sum_{z' \in V_K} \log(\mathcal{P}(c(z'))) \right\rceil} \cdot \sum_{z \in \Gamma(x)} \log(\mathcal{P}(c(z))), \tag{2}$$

where $\lceil \cdot \rceil$ is the ceiling operation that gives the smallest integer greater than the input, V_K is the vertex set of the enclosing subgraph to be labeled.

Now we prove that PALETTE-WL is color-order preserving.

THEOREM 3.4. *The WL algorithm with the hash function in (2) (PALETTE-WL)*

1. *has perfect hashing, i.e., $h(x) = h(y)$ if and only if: 1) $c(x) = c(y)$; and 2) $\Gamma(x)$ and $\Gamma(y)$ contain the same colors with the same cardinality; and*
2. *is color-order preserving.*

PROOF. We first prove the perfect hashing property, following a similar argument as in [13]. To prove the first direction ($h(x) = h(y) \Rightarrow$ two conditions), we see that $h(x) = h(y)$ means:

$$c(x) - c(y) = \frac{\sum_{z \in \Gamma(y)} \log(\mathcal{P}(c(z))) - \sum_{z \in \Gamma(x)} \log(\mathcal{P}(c(z)))}{\left\lceil \sum_{z' \in V_K} \log(\mathcal{P}(c(z'))) \right\rceil}. \tag{3}$$

We exponentiate both sides and write $N_x := \Pi_{z \in \Gamma(x)} \mathcal{P}(c(z))$, $N_y := \Pi_{z \in \Gamma(y)} \mathcal{P}(c(z))$, $Z := \left\lceil \sum_{z' \in V_K} \log(\mathcal{P}(c(z'))) \right\rceil$ (note that they are all integers). Then we have

$$e^{Z(c(x)-c(y))} = N_y / N_x. \tag{4}$$

On the left-hand side of the above equation, $Z(c(x) - c(y))$ is an integer. We know that all integral powers of e are irrational except for e^0. On the right-hand side, N_y/N_x is rational, which means that we must have $c(x) = c(y)$ (the first condition) and $e^{Z(c(x)-c(y))} = 1$. Thus we have $N_x = N_y$. Since N_x and N_y are integers, their prime factorizations must be the same. This implies that the two multisets $\{c(z)|z \in \Gamma(x)\}$ and $\{c(z)|z \in \Gamma(y)\}$ coincide (the second condition). This proves the first direction. The opposite direction can be easily proved using the definition of $h(x)$.

To prove that it is color-order preserving, we consider any pair of vertices x and y. Assume their colors at the i^{th} iteration have $c^i(x) < c^i(y)$. Note that $c^i(x)$ and $c^i(y)$ are integers, which implies that $c^i(x) + 1 \leq c^i(y)$.

Algorithm 3 The Palette-WL Algorithm

input: enclosing subgraph $G(V_K)$ centered at target link (x, y), which is extracted by Algorithm 2
output: final colors $c(v)$ for all $v \in V_K$
calculate $d(v) := \sqrt{d(v,x) \cdot d(v,y)}$ for all $v \in V_K$
get initial colors $c(\mathbf{v}) = f(d(\mathbf{v}))$
while $c(\mathbf{v})$ has not converged **do**
  calculate hashing values $h(v)$ for all $v \in V_K$ by (2)
  get updated colors $c(\mathbf{v}) = f(h(\mathbf{v}))$
end while[1]
return $c(\mathbf{v})$

We have:

$$h^i(x) = c^i(x) + \frac{1}{\left\lceil \sum_{z' \in V_K} \log(\mathcal{P}(c^i(z'))) \right\rceil} \cdot \sum_{z \in \Gamma(x)} \log(\mathcal{P}(c^i(z))) < c^i(x) + 1 \leq c^i(y) \leq h^i(y), \quad (5)$$

which means $h^i(x) < h^i(y)$. Therefore, in the next iteration we are guaranteed to have $c^{i+1}(x) < c^{i+1}(y)$. □

We call it Palette-WL because the labeling process is like to draw initial colors from a palette to vertices, and then iteratively refine them by mixing their original colors and nearby colors in such a way that the colors' relative ordering is preserved. We show the complete steps of Palette-WL in Algorithm 3. Vertices are first assigned initial colors according to their geometric mean distance to the vertices x and y of the target link. Then the initial colors are iteratively refined using the hash function in (2). To facilitate expression, we define $f : \mathbb{R}^K \to C^K$, which maps K real numbers to K colors. f first maps the smallest real number to color 1, and then maps the second smallest real number to color 2, and so on. If two or more real numbers are equal to each other, they are mapped to the same color. Such process is repeated until every real number is mapped to a color. We use $d(v_a, v_b)$ to denote the length of the shortest path between v_a and v_b.

Finally, we sort the vertices in V_K using their Palette-WL colors in ascending order. If there are vertices with the same color, we use Nauty, a graph canonization tool, to break the ties [20].

3.2.2 Represent enclosing subgraphs as adjacency matrices. Given an enclosing subgraph $G(V_K)$, Wlnm represents it as an upper triangular adjacency matrix whose vertex ordering is decided by V_K's Palette-WL colors. After that, the adjacency matrix is vertically read and input to a fully-connected neural network.

To further increase the flexibility of Wlnm, we can relax the 1/0 entries of adjacency matrix by letting them encode other information. In our experiments, we set $A_{i,j} = 1/d((i,j),(x,y))$ where $d((i,j),(x,y))$ is the length of the shortest path to reach link (i,j) from link (x,y).

3.3 Neural network learning

Training. After we encode the enclosing subgraphs, the next step in Wlnm is to train a classifier. To learn sophisticated nonlinear patterns, we resort to neural networks due to their unprecedented representation capability. For a given network $G = (V, E)$, we first construct the positive samples by selecting all edges $(x, y) \in E$. Then, we construct negative samples by randomly selecting $\alpha|E|$ pairs of $x, y \in V$ such that $(x, y) \notin E$. For a given training link (x, y) (positive or negative), Wlnm first extracts its enclosing subgraph and then encodes the enclosing subgraph into an adjacency matrix using the proposed Palette-WL algorithm. The adjacency matrices are vertically fed into a feedforward neural network together with their labels (1: $(x, y) \in E$, 0: $(x, y) \notin E$).

Note that the entry $A_{1,2}$ (shown as a star in Figure 4) should not be fed into the neural network, because it records the existence of the link (x, y). Although $A_{1,2}$ can be either 1 or 0 during training, $A_{1,2}$ is always 0 when we are predicting an unknown link. Adding this "class label" in our inputs will make the prediction on all testing links biased towards 0 (nonexistence).

Testing (link prediction). After training the neural network, we can predict the existence of a testing link by extracting its enclosing subgraph, encoding it using Palette-WL, and feeding the resulting adjacency matrix to the neural network. Finally, a prediction score between 0 and 1 is output for each testing link, which represents the estimated probability of the testing link being positive.

3.4 Discussions

The following analysis shows that 1) the colors generated by Palette-WL are guaranteed to converge; and 2) the adjacency matrices can be constructed efficiently.

Theorem 3.5. *If a color refinement algorithm is color-order preserving, then given any vertex v, its color is non-decreasing over iterations.*

Proof. Consider any vertex v whose color $c(v) = k$. We can prove the result by induction on k. When $k = 1$, v already has the smallest color, so its color is non-decreasing in the next iteration. Now consider the case when $c(v) = k + 1$ and, in the next iteration, its color is reduced to a color $l < k + 1$. Since every color in $\{1, 2, ..., k + 1\}$ has been assigned to at least one vertex, let v' be one such vertex whose color is l. By the induction hypothesis, v' will get a color larger than or equal to l, which contradicts the color-order preserving requirement. □

Lemma 3.6. *For a graph with K vertices, the Palette-WL algorithm takes at most K iterations to converge.*

Proof. Theorem 3.5 implies that the total number of colors does not decrease. Assume the algorithm has not converged at an iteration, there must be a vertex that has its current color c changed. By Theorem 3.5, it must increase its color. If c is already the largest color, increasing it will increase the color number. Otherwise, the vertex must increase c to some existing color c'. Due to the color-order preserving property, vertices that originally have color c' must increase their colors, too. Repeating this argument we see that finally the total number of colors must be increased. In either case, the color number increases at least by 1 in each iteration. Since there are at most K different colors in the end, the iteration number is bounded by K. □

In each iteration, we need to compute the hash function for each vertex by (2), which takes at most $O(K)$ time. Evaluating K hash functions needs $O(K^2)$ time (or $O(|E|)$ time if using sparse matrix-matrix multiplications, where E is the edge set of this graph). And the sorting needs $O(K \log K)$ time. Thus, the time complexity of each iteration is $O(K^2)$. Given Lemma 3.6, we have the following.

[1] There will be at most K iterations; see Lemma 3.6

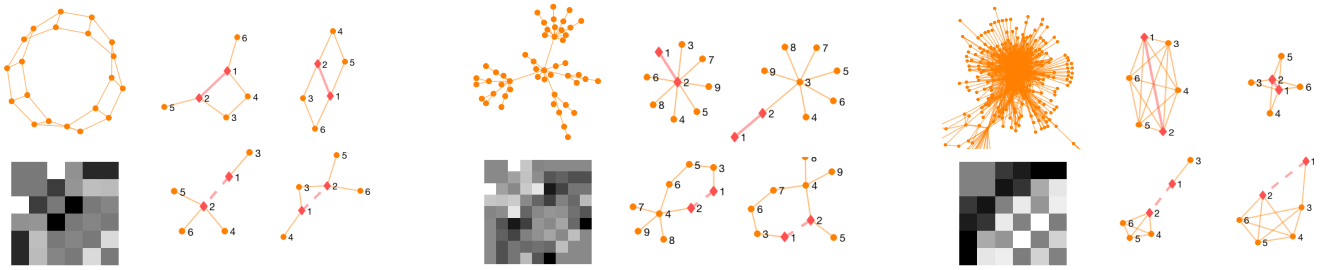


Figure 5: Visualization experiments for a 3-regular graph (left), a preferential attachment graph (middle), and a USAir network (right). In each block, the top left figure depicts the network, the bottom left visualizes the weight matrix trained on the network's enclosing subgraphs. Four example enclosing subgraphs are displayed beside them, including two most frequent positive enclosing subgraphs (the first row) and two randomly generated negative enclosing subgraphs (the second row). In each enclosing subgraph, two red diamonds correspond to the target link. The enclosing subgraph size K is set to 6, 9, 6 for the three networks, respectively.

Theorem 3.7. *For a graph with K vertices, the Palette-WL algorithm has $O(K^3)$ time complexity.*

The above results show that we can encode the enclosing subgraphs efficiently. We have three additional notes. First, after running Palette-WL, ties can be broken by running Nauty, which has an average time complexity of $O(K)$ [21]. Second, there exists $O((|E| + |V|) \log |V|)$ time implementation of WL using some special data structure [22]. However, the resulting WL is not color-order preserving. Third, subgraph pattern encoding is naturally parallelizable, which can further promote the scalability of Wlnm.

4 RELATED WORK

Link prediction [1] has been a hot topic for the past decade in data mining. Existing link prediction methods can be mainly categorized into two types: topological feature-based and latent feature-based. Topological feature-based methods predict links based on some local or global node similarity heuristics. Popular measures include common neighbors [1], Katz index [9], Adamic-Adar index [2], and PageRank [1]. We have surveyed them in Table 1. These heuristics do not perform well when the similarity scores do not capture the network's latent formation mechanisms. Latent feature-based methods predict links based on their latent features or latent groups, which can be extracted through low-rank decomposition of network's adjacency matrix [3], or trained by fitting some probabilistic models [5]. Popular latent feature-based methods include: matrix factorization [23]; ranking methods [24] which treat link prediction as a learning to rank problem; and stochastic block models [5, 25], which assume that nodes have latent groups and links are determined by the group memberships of nodes. Latent feature-based methods focus more on individual nodes than network topologies, thus cannot explain how networks are formed.

There has been research studying extracting local patterns from graphs to build graph kernels [26]. However, to the best of our knowledge, no existing research extracts subgraphs for link prediction. The use of graph labeling methods to impose a vertex ordering is introduced in [19]. In that paper, local subgraphs are extracted for nodes to define receptive fields around node pixels in order to learn a convolutional neural network for graph classification. Our paper extracts local subgraphs around links instead of node pixels, and our task is to predict the existence of links instead of classifying graphs. Moreover, we analyzed in-depth a particular graph labeling method, the Weisfeiler-Lehman algorithm, and proposed a new efficient and color-order preserving variant, Palette-WL, to meet the special requirements for link prediction learning. Graph labeling, especially WL, has also been used to design efficient graph kernels [18].

5 EXPERIMENTAL RESULTS

In this section, we conduct two types of experiments: a visualization on small datasets, and a performance comparison on real-world networks. All codes and datasets are publicly available at https://github.com/muhanzhang/LinkPrediction.

5.1 Visualization

We use two small artificial datasets and a real-world air line network [27] to visualize the learning ability of Wlnm. Here, for visualization purpose, we only train a logistic regression model on the enclosing subgraphs. Figure 5 depicts the networks, some extracted enclosing subgraphs, and the learned weights.

As we can see, Wlnm successfully extracts the building blocks for each network, and the weight patterns indicate how links in different networks are likely to form. For example, we observe that the most frequent positive enclosing subgraph of the USAir network is a clique. This makes sense since big cities tend to establish dense airline connections with other big cities too. The second most frequent one is also illustrative, as it depicts the pattern of four small cities connecting to two big cities.

We also display the Palette-WL labels for vertices of the enclosing subgraphs. As we can see, vertices 1 and 2 always correspond to the target link. Other vertices' labels characterize their structural roles within the subgraph and also preserve the enclosing subgraph's directionality.

Note that the visualized weights for the toy datasets are merely learned from a linear classifier. When the link formation mechanisms are complex, the need to learn sophisticated nonlinear features urges us to use neural networks in real-world experiments.

5.2 Experiments on real-world networks

To evaluate the performance of Wlnm, we compare it with 12 baselines on eight real-world networks.

Table 2: AUC results of 12 baseline methods, WLLR, and WLNM

Data	CN	Jac.	AA	RA	PA	Katz	RD	PR	SR	SBM	MF-c	MF-r	WLLR^{10}	WLNM^{10}	WLNM^{20}
USAir	0.940	0.903	0.950	0.956	0.894	0.931	0.898	0.944	0.782	0.944	0.918	0.849	0.896	**0.958**	**0.961**
NS	0.938	0.938	0.938	0.938	0.682	0.940	0.582	0.940	0.940	0.920	0.636	0.720	0.862	**0.984**	**0.981**
PB	0.919	0.873	0.922	0.923	0.901	0.928	0.883	0.935	0.773	0.938	0.930	**0.943**	0.827	0.933	0.939
Yeast	0.891	0.890	0.891	0.892	0.824	0.921	0.880	0.927	0.914	0.914	0.831	0.881	0.854	**0.956**	**0.951**
C.ele	0.848	0.792	0.864	0.868	0.755	0.864	0.740	**0.901**	0.760	0.867	0.832	0.844	0.803	0.859	0.854
Power	0.590	0.590	0.590	0.590	0.441	0.657	0.845	0.664	0.763	0.665	0.524	0.517	0.778	**0.848**	**0.874**
Router	0.561	0.561	0.561	0.561	0.471	0.378	0.926	0.380	0.367	0.857	0.779	0.783	0.897	**0.944**	0.915
E.coli	0.932	0.806	0.952	0.958	0.912	0.929	0.889	0.954	0.637	0.939	0.909	0.916	0.894	**0.971**	**0.976**
Ranking	7.875	10.625	7.500	6.875	12.875	7.125	10.375	5.125	11.000	5.625	10.500	9.500	10.125	2.500	2.375

Datasets. We use eight datasets: USAir, NS, PB, Yeast, C.ele, Power, Router, and E.coli. USAir is a network of US Air lines [27]. NS is a collaboration network of researchers who publish papers on network science [28]. PB is a network of US political blogs [29]. Yeast is a protein-protein interaction network in yeast [30]. C.ele is a neural network of C. elegans [31]. Power is an electrical grid of western US [31]. Router is a router-level Internet [32]. E.coli is a pairwise reaction network of metabolites in E. coli [33]. We include the dataset statistics in Table 3. In each dataset, all existing links are randomly split into a training set (90%) and a testing set (10%). Other potential edges are treated as unknown links. Area under the ROC curve (AUC) is adopted to measure the link prediction performance, which can be understood as the probability that a random positive link from the testing set has a higher score than a random unknown link.

Baselines and experimental setting. We compared WLNM with nine heuristic methods in Table 1. They are: common neighbors (CN), Jaccard (Jac.), Adamic-Adar (AA), resource allocation (RA), preferential attachment (PA), Katz, resistance distance (RD), PageRank (PR), and SimRank (SR). In addition, we also compared WLNM with three latent feature models: stochastic block model (SBM) [5], matrix factorization using a classification loss function (MF-c), and matrix factorization using a regression loss function (MF-r). For Katz, we set the damping factor β to 0.001. For PageRank, we set the damping factor d to 0.85 as suggested by [14]. For Katz and PageRank, we also tested $\beta = 0.01$ and $d = 0.7$. The results are very similar and thus not reported. For SBM, we use the implementation of [25]. For MF, we use the libFM [34] software. The number of latent groups of SBM is searched in $\{4, 6, 8, 10, 12\}$. The number of latent factors of MF is searched in $\{5, 10, 15, 20, 50\}$. The best result is reported for each dataset.

For the proposed WLNM, we report subgraph sizes $K = 10$ (WLNM^{10}) and $K = 20$ (WLNM^{20}). We randomly sample from the unknown links to construct negative examples. We set the number of sampled negative links to be twice of the given positive training links. The link prediction performance is evaluated on the testing set as well as a sampled set of unknown links which is also twice as large. The sampling is performed so that the training and testing links do not overlap. For the neural network structure, we use three fully-connected hidden layers with 32, 32, 16 hidden neurons respectively and a softmax layer as the output layer. Rectified linear unit (ReLU) is adopted as the activation function for all hidden layers. We adopt the Adam update rule [35] for optimization with a learning rate of 0.001 and a mini-batch size of 128. We set the number of training epochs to be 100. The model parameters with the best results on 10% validation splits of the training set are used to predict the testing links. The neural network is implemented using Torch [36]. To demonstrate the strength of neural networks, we also train a logistic regression model on the same enclosing subgraphs (under $K = 10$, we call this model WLLR^{10}).

Table 3: Comparison of different graph labeling methods (K=10). Palette-WL (PWL^c) performs the best on all datasets.

Data	$\lvert V\rvert$	$\lvert E\rvert$	PWL^c	PWL^1	HWL^c	Nauty	Rand
USAir	332	2126	**0.958**	0.777	0.758	0.767	0.607
NS	1589	2742	**0.984**	0.896	0.881	0.896	0.738
PB	1222	16714	**0.933**	0.730	0.726	0.725	0.609
Yeast	2375	11693	**0.956**	0.774	0.743	0.764	0.654
C.ele	297	2148	**0.859**	0.609	0.634	0.631	0.555
Power	4941	6594	**0.848**	0.647	0.665	0.641	0.550
Router	5022	6258	**0.944**	0.557	0.622	0.555	0.640
E.coli	1805	14660	**0.971**	0.863	0.857	0.838	0.773

All experiments are ran 10 times on a 12-core Linux server which has two NVIDIA TITAN GPUs with 6GB memory each. Under this configuration, 10 runs of WLNM on all datasets finish in 2.5 hours. The average AUC results are reported in Table 2.

Results. From Table 2, we can observe the following. WLNM generally performs much better than other baselines in terms of AUC. It outperforms all 12 baselines on USAir, NS, Yeast, Power, Router, and E.coli by a large margin. **Most remarkably**, WLNM performs very well on the two difficult datasets: Power and Router, on which most other methods can only perform slightly better than random guessing. This suggests that WLNM is able to learn "novel" topological features which current heuristics cannot express. Another interesting finding is that, for all the 12 baseline methods, none of them can perform well on all datasets. In comparison, WLNM performs consistently well—it has an AUC greater than 0.85 across all datasets, and often, the result is over 0.95. We also find that WLNM is robust under the variation of K. Its performance is similarly good for K=10, 20, 30, and 40 (not all reported).

To demonstrate the universality of WLNM, we calculate the rankings of all the methods based on AUC for each dataset, and append the average ranking of each method (over all datasets) to Table 2. Compared to other methods, WLNM shows substantial overall advantages, having the best average ranking of less than 2.5.

To further show the importance of Palette-WL in subgraph pattern encoding, we compare it to other four graph labeling methods under $K = 10$ in Table 3. Here, PWL^c denotes our Palette-WL method with initial distance-based colors. To show the usefulness of distance-based initial coloring, we compare Palette-WL with all vertices colored 1 initially PWL^1. To show the consequence of a WL that is not color-order preserving, we report the performance of HWL^c, which uses the hashing function in (1). As we can see, its performance is much worse than PWL^c, since it cannot preserve the relative ordering of vertices and thus resulting in chaotic final labelings. Finally, we also report the results by directly applying Nauty to get a canonical labeling for each subgraph (Nauty), and randomly ordering the vertices (Rand). We also did experiments using the classical WL algorithm and saw very similar results to PWL^c. However, on datasets PB and E.coli, WL cannot finish in 2 hours whereas Palette-WL finishes in minutes. Thus we do not list the results here. From Table 3, we can see that PWL^c outperforms all other variants by a large margin.

6 CONCLUSIONS

In this paper, we have proposed a next-generation link prediction method, Weisfeiler-Lehman Neural Machine (Wlnm), which learns topological features from networks by extracting links' local enclosing subgraphs. To properly encode a link's enclosing subgraph, we have proposed an efficient graph labeling algorithm called Palette-WL to impose an order on subgraph vertices based on their structural roles and the subgraph's intrinsic directionality. After that, a neural network is trained on the adjacency matrices to learn nonlinear topological features for link prediction. Experimental results have shown that Wlnm gives unprecedentedly strong performance compared to 12 state-of-the-art methods. Moreover, Wlnm exhibits great generality, i.e. the ability to automatically learn complex network topological features, as it performs consistently well across different networks.

ACKNOWLEDGMENTS

The authors would like to thank Roman Garnett, Sanmay Das, and Zhicheng Cui for the helpful discussions. The authors would also like to thank the anonymous reviewers for their valuable comments. The work is supported in part by the DBI-1356669, SCH-1343896, III-1526012, and SCH-1622678 grants from the National Science Foundation of the United States.

REFERENCES

[1] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.
[2] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.
[3] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.
[4] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *arXiv preprint arXiv:1503.00759*, 2015.
[5] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9 (Sep):1981–2014, 2008.
[6] Tolutola Oyetunde, Muhan Zhang, Yixin Chen, Yinjie Tang, and Cynthia Lo. Boostgapfill: Improving the fidelity of metabolic network reconstructions through integrated constraint and pattern-based methods. *Bioinformatics*, 2016.
[7] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning (ICML)*, pages 880–887. ACM, 2008.
[8] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71(4):623–630, 2009.
[9] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.
[10] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011.
[11] Douglas J Klein and Milan Randić. Resistance distance. *Journal of Mathematical Chemistry*, 12(1):81–95, 1993.
[12] Boris Weisfeiler and AA Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2 (9):12–16, 1968.
[13] Kristian Kersting, Martin Mladenov, Roman Garnett, and Martin Grohe. Power iterated color refinement. In *AAAI*, pages 1904–1910, 2014.
[14] Sergey Brin and Lawrence Page. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer networks*, 56(18):3825–3833, 2012.
[15] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
[16] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543. ACM, 2002.
[17] William Cukierski, Benjamin Hamner, and Bo Yang. Graph-based features for supervised link prediction. In *The 2011 International Joint Conference on Neural Networks (IJCNN)*, pages 1237–1244. IEEE, 2011.
[18] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.
[19] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd annual international conference on machine learning. ACM*, 2016.
[20] Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, 60(0):94 – 112, 2014. ISSN 0747-7171.
[21] László Babai, Paul ErdoËĬ s, and Stanley M Selkow. Random graph isomorphism. *SIAM Journal on Computing*, 9(3):628–635, 1980.
[22] Christoph Berkholz, Paul Bonsma, and Martin Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. In *European Symposium on Algorithms*, pages 145–156. Springer, 2013.
[23] Kurt Miller, Michael I Jordan, and Thomas L Griffiths. Nonparametric latent feature models for link prediction. In *Advances in neural information processing systems*, pages 1276–1284, 2009.
[24] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.
[25] Christopher Aicher, Abigail Z Jacobs, and Aaron Clauset. Learning latent block structure in weighted networks. *Journal of Complex Networks*, 3(2):221–248, 2015.
[26] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.
[27] Vladimir Batagelj and Andrej Mrvar. http://vlado.fmf.uni-lj.si/pub/networks/data/, 2006.
[28] Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.
[29] Robert Ackland et al. Mapping the us political blogosphere: Are conservative bloggers more prominent? In *BlogTalk Downunder 2005 Conference, Sydney*. BlogTalk Downunder 2005 Conference, Sydney, 2005.
[30] Christian Von Mering, Roland Krause, Berend Snel, Michael Cornell, Stephen G Oliver, Stanley Fields, and Peer Bork. Comparative assessment of large-scale data sets of protein–protein interactions. *Nature*, 417(6887):399–403, 2002.
[31] Duncan J Watts and Steven H Strogatz. Collective dynamics of âĂŸsmall-worldâĂŹnetworks. *Nature*, 393(6684):440–442, 1998.
[32] Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring isp topologies with rocketfuel. *IEEE/ACM Transactions on networking*, 12(1):2–16, 2004.
[33] Muhan Zhang, Zhicheng Cui, Tolutola Oyetunde, Yinjie Tang, and Yixin Chen. Recovering metabolic networks using a novel hyperlink prediction method. *arXiv preprint arXiv:1610.06941*, 2016.
[34] Steffen Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.
[35] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 41–48. ACM, 2009.
[36] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.