Applied Soft Computing 120 (2022) 108657

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

LWP-WL: Link weight prediction based on CNNs and the Weisfeiler–Lehman algorithm

Unai Zulaika *, Rubén Sánchez-Corcuera, Aitor Almeida, Diego López-de-Ipiña

DeustoTech, University of Deusto, Avda. Universidades, 24 Bilbao (Bizkaia) 48007, Spain

ARTICLE INFO

Article history:
Received 1 July 2020
Received in revised form 11 February 2022
Accepted 20 February 2022
Available online 28 February 2022

Keywords:
Link weight prediction
Graph mining
Weisfeiler–Lehman algorithm

ABSTRACT

We present a new technique for link weight prediction, the Link Weight Prediction Weisfeiler–Lehman (LWP-WL) method that learns from graph structure features and link relationship patterns. Inspired by the Weisfeiler–Lehman Neural Machine, LWP-WL extracts an enclosing subgraph for the target link and applies a graph labelling algorithm for weighted graphs to provide an ordered subgraph adjacency matrix into a neural network. The neural network contains a Convolutional Neural Network in the first layer that applies special filters adapted to the input graph representation. An extensive evaluation is provided that demonstrates an improvement over the state-of-the-art methods in several weighted graphs. Furthermore, we conduct an ablation study to show how adding different features to our approach improves our technique's performance. Finally, we also perform a study on the complexity and scalability of our algorithm. Unlike other approaches, LWP-WL does not rely on a specific graph heuristic and can perform well in different kinds of graphs.

1. Introduction

Link prediction [1] is an important task in graph mining. Due to the wide use of networks, it has several applications such as friend [2] or item [3,4] recommendation, knowledge graph completion [5], temporal networks [6] and metabolic network reconstruction [7,8].

Novel techniques such as stochastic block models [9] or probabilistic matrix factorisation [10] have recently been proposed for link prediction. Before, simple yet powerful approaches based on heuristic methods were the most effective for solving this task. Heuristics are features that belong to the graph structure - i.e. those features correspond to the observed node and edge structures of the graph - and therefore can be obtained directly from the graph. These methods rely on similarity metrics between nodes for making predictions on links, and despite their simplicity, they provide high performances on social networks [11]. While those heuristics have good results locally, they do not generalise. Thus, they cannot be applied in various network types. Furthermore, high-order heuristics that perform better than first and second-order ones sometimes demand a considerable amount of resources to be computed. Ideally, we would want a model to automatically learn those heuristics from the graph.

Recent deep learning-based approaches that tackle this problem have been proposed recently. Weisfeiler–Lehman Neural Machine (WLNM) [12] is a method for link prediction that extracts enclosing subgraphs from possible links between nodes. Then, it does feed the flattened adjacency matrix for the enclosed subgraph into a neural network that predicts the existence of the link. SEAL (Subgraphs, Embeddings, and Attributes for Link prediction) [13], is a link prediction framework for learning from local enclosing subgraphs, embeddings and attributes based on graph neural networks.

However, these approaches do not provide a solution for weighted networks (networks that contain weights on edges). Link weight prediction is a subtask of link prediction which focuses on predicting weight values for linked entities in networks. It is applicable in weighted graphs such as collaboration networks [14], transport networks [15] or protein interaction networks [16]. Unfortunately, there are few deep learning approaches for solving link weight prediction problems. ModelR [17], which improves previous techniques like Stochastic Block Model, is the current state-of-the-art method. Notably, Node2vec [18], is an unsupervised random walk approach for developing representations of nodes that can be used for other tasks such as link weight prediction. Graph Convolutional Networks (GCNs) [19], is a graph specific model that applies convolutions over graph nodes to generate a representation for every node in the graph. We follow recent approaches using subgraph extraction for our method to propose a novel weighted link prediction framework, which achieves state-of-the-art performance on the task.

* Corresponding author.
E-mail address: unai.zulaika@deusto.es (U. Zulaika).

https://doi.org/10.1016/j.asoc.2022.108657

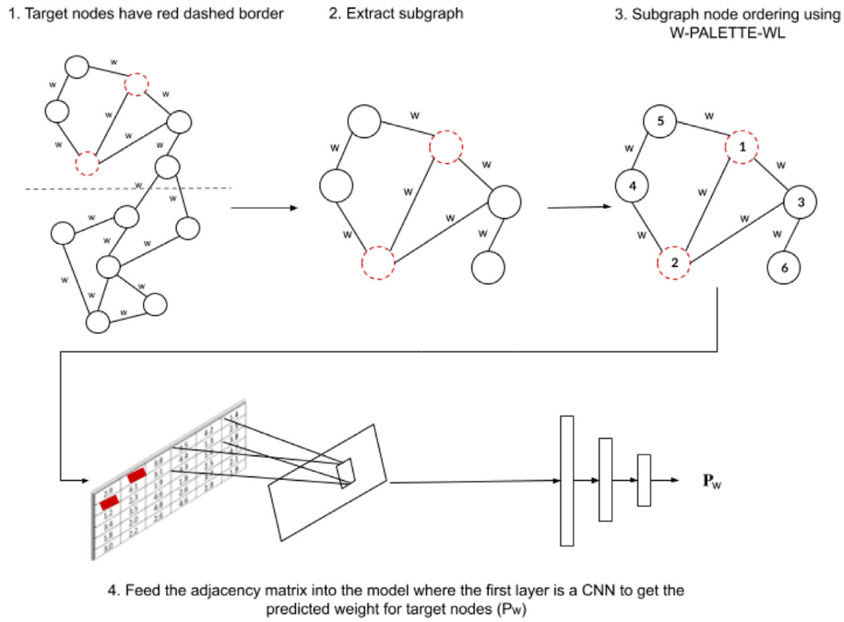

Fig. 1. Summary of the steps for the LWP-WL link weight prediction framework.

Inspired by the WLNM, we develop a new link weight prediction approach named **L**inked **W**eight **P**rediction **W**eisfeiler-**L**ehman, LWP-WL. The approach first extracts an enclosing subgraph from the neighbourhood of the target link. Then, the enclosing subgraph is fed into a neural network model (where the first layer is a Convolutional Neural Network (CNN)) that predicts the output value for the target link. The network is trained on target links to learn a link weight prediction model. The framework is depicted in Fig. 1.

The most remarkable features of our contribution can be identified as **(1)** a novel weight link prediction framework that learns from graph structure features **(2)** an extension of the graph labelling method WL (Weisfeiler–Lehman) for weighted networks in order to encode enclosing subgraphs into adjacency matrices so that neural networks can learn meaningful patterns **(3)** a novel edge to edge and edge to node filters applied to the model's CNN that learns patterns between adjacency matrices' values and **(4)** a collection of experiments on several datasets comparing LWP-WL and different techniques as well as an ablation study. With LWP-WL, we can achieve state-of-the-art results on various datasets, as demonstrated in the experiments section. The source code has been released on github.[1]

2. Related work

Link weight prediction [17,20–22] has been addressed in several previous works in the literature. For years, all of the approaches for this task relied on graph heuristics and latent features. Many works based on heuristics used node similarity [23,24], proximity measures [25] or local rankings [26] to solve the task, however different alternative measures could be used, e.g. PageRank [27]. Methods relying on latent features such as Stochastic Block Model [28,29] and matrix factorisation [30,31] also were presented. The models based on Stochastic Block Models and their variants were developed as generative models to find communities in the graph. Unlike the model presented in this paper, they are limited by design to perform link weight prediction since they find groups of nodes belonging to a community.

However, there was no approach based on deep learning for link weight prediction until recently. ModelR [17] provided a novel deep learning-based technique for the task. This approach relies on a neural network that receives the target nodes' ids as input, maps those ids to a feature vector and passes them through two fully-connected hidden layers to an output layer. Althought ModelR is the most similar method to the presented in this paper, our model applies a series of techniques that make it more advanced. The generalisation power of ModelR is limited by the randomness of the node ordering process, i.e. the nodes of the graph are assigned arbitrarily, and thus there is no consistency between different graphs when applying ModelR. Our method applies a novel graph labelling technique to overcome this issue. Furthermore, our method is more efficient and scalable than ModelR.

Nevertheless, graph-related works appeared in the literature in recent years [32]. Besides factorisation-based methods, two main approaches are considered in the literature: random-walk based techniques and Graph Neural Networks (GNNs). Random-walk based techniques [18,33] rely on performing random walks on the graph to generate node representations that are similar between each other when nodes co-occur on those random walks. In this work, we compare our model to node2vec [18] which is a widely used approach when compared to random-walk based methods in the literature. On the other hand, GNNs [34] are networks capable of performing convolutional operations on non-Euclidean data domains. We compare our approach to GCNs [19] that are one of the most robust approaches in the field of GNNs. GCNs and our method share characteristics that make them comparable: (a) they both apply a structural role to the nodes by using the WL algorithm, and (b) they both use convolutions to aggregate the nodes information. The main difference between them relies on how they perform the convolution. GCNs do a neighbourhood aggregation, while our method uses special filters for a CNN (Convolutional Neural Network) layer (further explained in Section 4. In this manner, our method aggregates information in a more meaningful process for the link weight prediction task.

Various research studies have applied graph labelling algorithms for different graph-related tasks. Graph labelling was introduced as a technique in [35] which presented an approach for

[1] https://github.com/unai-zulaika/LWP-WL-Pytorch.

learning a convolutional neural network for graph classification. Graph labelling has also been recently used for link prediction. The Weisfeiler–Lehman Neural Machine [12] which inspired this work, proposed an algorithm based on the original WL algorithm for labelling non-weighted graphs. The SEAL framework [13] also presented a novel node labelling method based on the radius from each node to the target link. These works are focused on simple link prediction, i.e. they are not suited for weights. Their node labelling algorithm cannot handle weighted nodes and, thus, does not apply to this work's task. Furthermore, the method we present uses CNNs with special filters to further enhance the weight prediction while Weisfeiler–Lehman Neural Machine and SEAL use other techniques.

Few approaches in the literature have used Convolutional Neural Networks over graph data, none that we know for link weight prediction. Due to the non-grid structure of graphs, these kinds of networks are not suitable for graphs. However, successful approaches have used CNNs for other graph-related tasks. ConvE [36] presented an approach for link prediction on knowledge graphs that applies a convolution over 2D shaped embeddings of nodes. ConvKB [37] also employs a CNN to capture global relationships and transitional characteristics between entities and relations in knowledge bases and perform link prediction. However, these models were conceived for multi-relational data and cannot be applied for the link weight prediction task. Remarkably, they served as an inspiration for this work.

An in-depth comparison to the most notable models for link weight prediction is provided in Section 5.

3. Background

This section provides the background needed to understand the LWP-WL approach.

3.1. Graphs

An undirected weighted graph G can be denoted as $G = (V, E)$ where $V = \{v_1, v_2, \dots, v_n\}$ (being N the number of vertices or nodes) is the set of nodes and $E \subseteq V \times V$ is the set of edges (or links). Each link contains an associated weight value w, so we represent a given link as $e_{v1,v2} = (v_1, v_2, w_{v_1,v_2})$. The graph can be represented by an adjacency matrix A, where $A_{i,j} = w_{v_i,v_j}$ if there is an link from i to j and $A_{i,j} = 0$ otherwise. Additionally, the distance between two connected nodes is $\sum_{i',j'}^{N'} d(v_{i'}, v_{j'})$ where $v \in V'$ and V' is the set of nodes for the shortest path from the origin to the target node.

3.2. Weisfeiler-Lehman algorithm

The Weisfeiler–Lehman (WL) algorithm [38] is a method to solve the graph isomorphism problem, i.e. whether two graphs contain the same number of nodes connected in the same way.

In the present work, we use the 1-dimensional WL algorithm as a **graph labelling** method (also called graph colouring). Graph labelling is a function that maps a set of nodes to an ordered set,

$$l : V \rightarrow O \tag{1}$$

where V is the set of nodes and O is the ordered set of nodes.

WL does iteratively update node labels using their neighbours' labels and compresses the updated labels into new labels until convergence. At first, all nodes are set to the same colour, i.e. 1. For each node, it gets a **signature string** by concatenating its colour and the sorted colours of its neighbours. Nodes are then sorted by the ascending order of their signature strings and assigned new colours. Nodes with the same signature strings get the same colour. The algorithm is shown in Algorithm 1.

Algorithm 1: Weisfeiler–Lehman algorithm [38].

```
input: graph G = (V, E), initial colours c^0(v) = 1 for all
  v ∈ V, γ(v) denotes neighbours set for v
output: final colours c(v) for all v ∈ V
let c(v) = c^0(v) for all v ∈ V
while c(v) has not converged do
    for each v ∈ V do
        collect the set of colours {c(v')|v' ∈ γ(v)} for
          neighbours
        sort the set in ascending order
        concatenate the sorted set to c(v) to generate a
          signature string s(v) = ⟨c(v), {c(v')|v' ∈ γ(v)}sort⟩
    end
    sort all s(v) in lexicographical ascending order
    map all s(v) to new colours sequentially, same strings
      get the same colour
end
```

An essential characteristic of WL algorithm is that it does encode **structural roles** from vertices inside the graph. Additionally, it defines a relative ordering for those vertices taking their structural role into account; vertices that play the same role are similarly labelled. Finally, it is remarkable that this relative ordering for vertices is consistent across different graphs.

4. Proposed approach

In this section, we introduce our approach for link weight prediction, LWP-WL. It does contain the following steps:

1 Subgraph extraction.
2 Subgraph node ordering.
3 Feeding the ordered subgraph's adjacency matrix into the model.

A summary figure of the whole approach is shown in Fig. 1.

4.1. Subgraph extraction

The first step of the approach is to extract enclosing subgraphs for target links. We identify the target nodes of a target link as the pair of nodes that perform the connection for the specific link.

For the target link, e_{v_x,v_y}, we extract a set of nodes of a user-defined size K. Starting from the nodes that hold the target link - i.e. v_x and v_y - we apply the Breadth-First-Search (BFS) [39] strategy to add nodes to the subgraph. Although we opted for BFS, other strategies can be applied. In this way, we start adding nodes from the 1-hop neighbourhood of the target nodes and then we iteratively continue to the next neighbourhood until we reach the number of nodes K and finally get the set of nodes for the subgraph, L_K. In order to obtain the representation for the target link, we add the same amount of nodes for each of the target nodes, so we apply $BFS(v_x)$ to get a set of nodes L_{v_x} of size $K/2$ and $BFS(v_x)$ to get a set of nodes L_{v_y} of size $K/2$. Finally, we combine both sets $L_{v_x} \cup L_{v_y}$ to get the final set L_K. We ensure that nodes from both sets are different so $L_{v_x} \cap L_{v_y} = \emptyset$. We provide the algorithm for the process in Algorithm 2.

The enclosing subgraph extraction process provides two remarkable additional features to the LWP-WL approach:

1. It can extract a **similar context representation for all links**, even from different graphs. By applying the same strategy for the subgraph extraction on every link, the

model will, later on, be able to generalise between different graphs, nodes and links.
2. It is scalable. The extraction of a fixed size K subgraph for every target link **avoids the need to input a representation for the whole graph** such as the adjacency matrix. In this manner, the input to the LWP-WL approach will always be of size $K \times K$ while avoiding inputs of size $V \times V$, which are not scalable as they grow with the number of nodes within the graph.

Algorithm 2: Subgraph extraction process. *BFS* stands for breadth-first search. $\Gamma(n)$ stands for n-hop neighbourhood.

```
input: graph G = (V, E), target nodes (x, y), subgraph size K
output: subgraph G(L_K) for (x, y)
L_k = []
L_x = [v_x]
L_y = [v_y]
n = 1
while |L_k| < K do
    for node in L_x do
        neighbourhood = BFS_n(node)
        for neighbour in neighbourhood do
            if |L_x| < K/2 then
                add neighbour to L_x
            end
        end
    end
    for node in L_y do
        neighbourhood = BFS_n(node)
        for neighbour in neighbourhood do
            if |L_y| < K/2 then
                add neighbour to L_y
            end
        end
    end
    n = n + 1
    L_k = L_x ∪ L_y
end
return subgraph G(L_K)
```

4.2. Subgraph node ordering

The second step of the approach is to order the extracted enclosing subgraph. So, we need a function that orders the set of nodes extracted from the enclosing subgraph.

$$l : L_K \rightarrow O_K \quad (2)$$

The purpose of the ordering is to provide a **consistent manner of labelling nodes** in a way that those nodes that have similar topological features concerning the subgraph are labelled similarly. Then, we can train a model that receives as input a representation of the ordered subgraph, namely, the adjacency matrix.

We adapt the WL graph labelling method to apply it for weighted graphs using the 1-dimensional Weisfeiler–Lehman (WL) algorithm. Authors from [12] also developed a node ordering algorithm based on WL. However, their approach was not developed for weighted graphs. We now mention the requirements for the graph labelling algorithm as they did in their work:

1. The graph labelling algorithm must provide **similar labels to nodes that have similar topological features** within the enclosing subgraph.
2. It must preserve the **topological directionality towards the target link**, i.e. the order of the nodes has to be conditioned by the target nodes, and the distance to them must be reflected in the ordering.

Both requirements are needed for developing models that use the nodes ordering for learning. We adapted the algorithm for weighted graphs because the WL method does not fulfil the second requirement.

We propose Weighted-WL (W-WL), a graph labelling algorithm whose objective is to order the set of nodes from the extracted subgraph, L_K from 1 to K. However, as we want to keep the topological directionality towards the target link, the target edges will always be given orders 1 and 2. First, nodes are assigned initial colours (rankings) according to the sum of their shortest paths to the target nodes, v_x and v_y (edges' weights are computed to calculate the shortest path). Afterwards, we assign a **signature string** to each of the nodes by using the Weisfeiler–Lehman algorithm, s_v. For this, we concatenate the initial colour of each node with the initial colours of its neighbours lexicographically ordered from minimum to maximum, generating a unique signature string. Then, the **lexicographically lowest string signature** corresponds to the next node for the ordered list. Finally, we iterate this process until every node is assigned a number. Because we have considered weights, it is unlikely to have ties in signature strings; otherwise, they are solved by choosing randomly. The process is defined in Algorithm 3. For a figure depicting the first steps of the process, see Fig. 2.

Once the node ordering process has finished and we have obtained the ordered set, we extract the adjacency matrix for the subgraph, $A_{s(O_K)}$, so the matrix's rows and columns correspond to the ordered set. This adjacency matrix is then used as input to a neural network. Before feeding it, we make sure that the values of the matrix that represent the weight for the link are not visible to the model, $A_{1,2}$ and $A_{2,1}$ as they are the labels for the model.

Algorithm 3: The W-WL Algorithm.

```
input: enclosing subgraph G(L_K) centred at target link e_{v_x,v_y},
  which is extracted by Algorithm 2
output: ordered set of nodes (O_K) from v ∈ L_K
L_1 = x, L_2 = y
calculate d(v) := d(v, x) + d(v, y) for all v ∈ L_K
get initial colours c(v) = f(d(v))
while |O_K| < |L_K| do
    calculate signature strings s(v) for all v ∈ L_K
    add lowest(s(v)) to O_K
end
return: O_K
```

4.3. Model

After obtaining the subgraph and the list of ordered nodes, we get an adjacency matrix $A(O_K)$; the next step is to train a classifier. For this classifier, we use a neural network.

As our main contribution, we use a Convolutional Neural (CNN) in the first layer of the neural network. This layer applies special filters adapted to the graph representation used, the adjacency matrix.

4.3.1. Edge to edge filter

For a given edge e_{v_x,v_y} this filter applies a convolution over edges that are neighbours of both v_x and v_y. In this manner, the filter learns about the edge's neighbourhood and the patterns among the nodes.

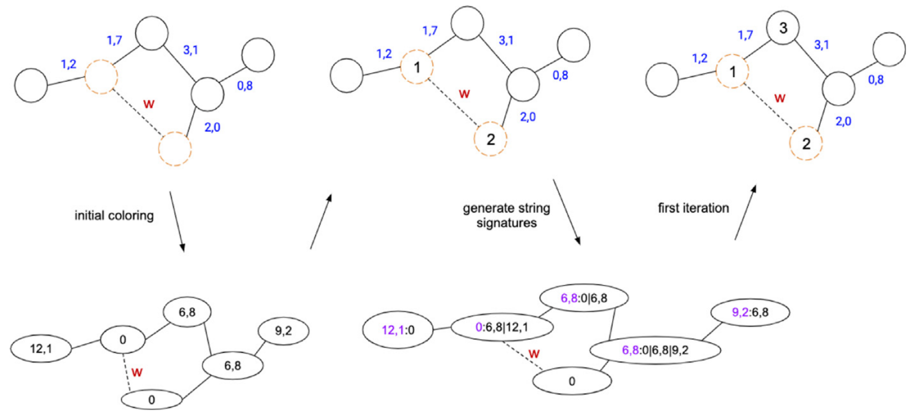

Fig. 2. First iteration of the W-WL algorithm. We want to predict the weight (w, coloured in red) of the link for the target nodes (dashed and coloured in yellow). First, we generate initial colours for each node. Then we generate a string signature for the nodes. Finally, we evaluate all string signatures and select the lexicographically lowest one. Target nodes get the 1 and 2 numbers to preserve the topological directionality.

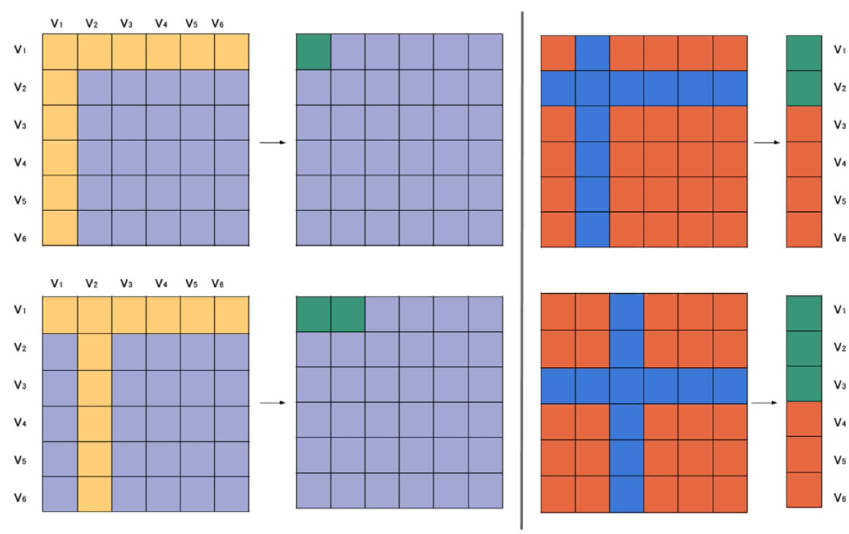

Fig. 3. The special filters applied to a $K = 6$ size adjacency matrix. **Left**: edge to edge filter applies a convolution over the neighbouring edges for the target link. **Right:** edge to node filter that applies a convolution over the neighbouring edges of each node in the adjacency matrix.

4.3.2. Edge to node filter

For a given node v_x, we apply a convolution for every neighbour edge of the node, unlike the edge to edge filter; it does focus on nodes learning patterns for their neighbourhood.

A figure depicting these filters is shown in Fig. 3. Then, the output of the CNN is flattened and goes through a set of fully-connected hidden layers to an output layer that performs a linear regression for the predicted weight.

For training a graph $G = (V, E)$, we build the samples by selecting all the links and their weights $e, w_e \in E$. Then, we divide those samples into training, validation and test sets. Next, we extract all subgraphs for each edge sample, and we train the system by providing as input the ordered adjacency matrix of the enclosing subgraph for the target link, $A_e(O_K)$, and the weight of that edge as the output w_e.

5. Experiments

In this section, we conduct a set of experiments to evaluate the performance of the LWP-WL approach. We first describe the datasets used for the experiments. Then, we provide an overview of the setup for those experiments. Next, we run a set of experiments comparing our approach with several methods, including the state-of-the-art, ModelR, and address the scalability and complexity aspects of our approach. Finally, an ablation study is presented, demonstrating how gradual enhancements to the LWP-WL approach impact the performance.

The conducted experiments aim to solve the next questions:

1. How well does LWP-WL perform in the link weight prediction task compared to the state-of-the-art models? Do the best performing models share any common characteristics?
2. How scalable is LWP-WL compared to the best models? Does it perform well in big graphs?
3. What is the impact of structural roles of nodes and, thus, the Weighted-WL graph labelling algorithm? And the special edge-to-edge and edge-to-node filters?

Each of those corresponding to the following subsections: *Comparison*, *Scalability and Complexity* and *Ablation Study*.

5.1. Datasets and baselines

In this work, we have used six networks to evaluate our model. We first select the standard four datasets used for link weight prediction included the state-of-the-art approach [17] to perform a comparison. Additionally, we also add two extensive networks further to analyse the performance of our model in big graphs.

Table 1
Summary of the datasets used for experiments, $|V|$ represents the number of nodes and $|E|$ the number of links.

Name	$\lvert V \rvert$	$\lvert E \rvert$
Astro	16,706	121,251
Airport	500	5960
Collaboration	226	20,616
Congress	163	26,569
Geom	7343	11,898
Forum	1899	20,291

These weighted graphs are real-world networks that belong to different domains and applications and report significant values for the task.

- *Airport* [40] contains the busiest airports in the US where nodes represent airports, links connections between them and weights correspond to the number of passengers travelling from one airport to the other.
- *Collaboration* [41] represents world nations as nodes, links and their weights are the number of papers written by authors from the two nations.
- *Congress* [42] is the network of the 102nd US Congress committees where nodes represent members and links, and their weights interlock value of shared members from the two committees.
- *Forum* [43] is a student social network where nodes are users and links and weights the number of messages sent from one student to the other.
- *Geom* [44] authors collaboration network in computational geometry obtained from the Computational Geometry Database geombib.
- *Astro* [45] a network of coauthorships between scientists posting preprints on the Astrophysics E-Print Archive.

As shown in the dataset statistics summary in Table 1, the six datasets are different in the number of vertices and edges and provide different values in their weights, providing a variety of settings for testing different models and performances.

Different baselines that have been proposed in the literature for this task are evaluated with these datasets:

- SBM (Stochastic Block Model) [46]: conceived for graph clustering in unweighted graphs, this approach partitions nodes into L subsets $C_i \subset V$ where $i \in L \in \mathbb{N}$.
  Then, SBM fits parameters θ for building the adjacency matrix A where $A_{ij} \sim B(1, \theta_{C,C_j})$ follows a Bernoulli distribution:
  $$P(A|C,\theta) = \prod_{ij} \theta_{C_i,C_j}^{A_{ij}} (1-\theta_{C_i,C_j})^{1-A_{ij}} \tag{3}$$
  , the log likelihood of A can be rewritten as an exponential family:
  $$\log P(A|C,\theta) = \sum_{ij} T(A_{ij}) \eta(\theta_{C,C_j}) \tag{4}$$
  where $T(A_{ij}) = (A_{ij}, 1)$ is the vector function of sufficient statistics of the Bernoulli random variable and
  $$\eta(\theta) = \left(\log\left(\frac{\theta}{1-\theta}\right), \log(1-\theta)\right) \tag{5}$$
  is the vector function of natural parameters of the Bernoulli random variable.
  Stochastic Block Models are generative models meant to find community structures in graph data. Therefore, they are limited by design to perform link weight prediction. While LWP-WL is a model conceived specifically to predict weights in graphs, SBMs in their standard version do exact recovery techniques without taking the link's weights into account.
- pWSBM (pure Weighted Stochastic Block Model) [29]: unlike SBM, it only uses link weight information. The main difference is that the adjacency matrix follows a normal distribution in its values, therefore $A_{ij} \sim N(\mu_{c,c_j}, \sigma^2_{c,c_j})$ and θ_{C,C_j} becomes the weight distribution parameter:
  $$\theta_{C,C_j} = (\mu_{C,C_j}, \sigma^2_{c,c_j}) \tag{6}$$
  and $T(A_{ij}) = (A_{ij}, A_{ij}^2, 1)$ becomes the vector function of sufficient statistics of the normal random variable, and
  $$\eta(\theta) = \left(\frac{\mu}{\sigma^2}, -\frac{1}{2\sigma^2}, -\frac{\mu^2}{2\sigma^2}\right) \tag{7}$$
  is the vector function of the natural parameters of the normal random variable. Then, it fits by
  $$\log P(A|C,\theta) = \sum_{ij} \left(A_{ij} \frac{\mu_{C,C_j}}{\sigma^2_{C,C_j}} - A_{ij}^2 \frac{1}{2\sigma^2_{C,C_j}} - \frac{\mu^2_{C,C_j}}{\sigma^2_{C,C_j}} \right) \tag{8}$$
  pWSBM is a generalised version of SBM for weighted graphs. Although it shares the same strengths, it is suited for link weight prediction. The idea behind pWSBM is to find the latent structural roles of the nodes in the graph. To do so, the WSBM models each weighted edge as a draw from a parametric exponential family distribution and recovers the whole graph and its weights. Our model, LWP-WL, conceives the same idea of finding latent structural roles in the graph more sophisticatedly. First, it finds the structural role by applying the W-WL labelling algorithm, which uses the nodes' topological features. It is essential to realise that W-WL does not try to fit each node into a block structure, so it is more flexible when finding latent structures. Second, the edge to edge and edge to node filters in the LWP-WL model allows predicting weights at an individual level for each of the edges, while the WSBM depends on group memberships and their parameters for the prediction.
- bWSBM (balanced Weighted Stochastic Block Model) [29]: it does mix the SBM and pWSBM approaches by using the pair (T_e, η_e) which denotes the family of link existence distributions in SBM and the pair (T_w, η_w) that denotes the family of the link weight distributions in pWSBM and $\alpha \in \mathbb{R} \in [0, 1]$ is a hyperparameter for their relative importance, E is the set of observed interactions, and W is the set of weighted edges. Thus, the fitting follows:
  $$\log P(A|C,\theta) = \sum_{ij} \left(T_e(A_{ij})\eta(\theta_{C,C_j})\right) + (1-\alpha) \sum_{ij \in W} \left(T_w(A_{ij})\eta_w(\theta_{C,C_z})\right) \tag{9}$$
  As bWSBM is a hybrid version of SBM and pWSBM, it also shares its differences with our model. The final purpose of block models is to find communities by recovering the graph. Balancing the characteristics of SBM and pWSBM does not provide any benefit over LWP-WL, and our model overperforms bWSBM for the same reasons stated before.
- DCWBM (Degree Corrected Weighted Stochastic Block Model) [29]: it incorporates node degree by replacing pair (T_e, η_e) in the bWSBM with:
  $$T_e(A_{ij}) = (A_{ij}, -d_i d_j) \tag{10}$$
  and
  $$\eta_e(\theta) = (log\theta, \theta) \tag{11}$$
  where d_i is the degree of node i.

Adding the node degree to the SBM is a method for explicitly incorporating more information into the model. However, DCWBM has less accurate predictions in predicting edge existence and edge weight (see Table 2). Furthermore, as pointed out in [29], degree correction methods perform better when doing community detection tasks. In LWP-WL, node degrees are not explicitly added to the model. Nevertheless, their structural roles are considered when performing the W-WL ordering algorithm. While explicitly adding node degree information to models does not improve performance in the link weight prediction task, information obtained from structural roles does (as the ablation study presented in this paper concludes).

- ModelR [17]: is a deep learning approach for link weight prediction. It follows a simple embedding approach with a node mapping layer that maps a given node (by id) to its vector representation. Then, multiple fully hidden connected hidden layers of rectified linear units lead to an output layer which performs a linear regression:

$$f : \{n_{id\_i}, n_{id\_j}\} \rightarrow w_{ij} \tag{12}$$

where $\{n_{id\_i}, n_{id\_j}\}$ are the ids for nodes i and j and $w_{ij} \in \mathbb{R}$ is the predicted weight for their link. The vector representations are updated via backpropagation (SGD is chosen for optimisation). It is the state-of-the-art approach for link weight prediction. ModelR is the most comparable method with LWP-WL. Both models use deep learning to solve the task. However, LWP-WL applies a series of techniques that make it more advanced. ModelR follows a simple schema where each node is assigned to a representation, and then a set of operations are applied through multiple fully hidden layers to end up on a weight prediction. This procedure limits the generalisation power of the model due to the randomness of the node ordering process, i.e. the nodes are assigned arbitrarily, and thus there is no consistency between different graphs. Additionally, by giving each node a representation, the scalability of the approach is linearly limited by the number of nodes in the graph, making the method unsuitable for large graphs. LWP-WL does solve this by applying the node ordering algorithm and extracting subgraphs to make the approach's scalability independent of the number of nodes in the graph. Furthermore, the application of CNNs and the special filters presented in this work correspond to an improvement over the fully connected layers of ModelR.

- Node2vec [18]: is a method that learns node embeddings based on random walk statistics. The difference between node2vec and other random walk approaches relies on its random walk producing strategy. It does bias the random walk using two parameters: p and q. These parameters control how the random walk explores the graph, concretely how exploring and leaving the neighbourhood of the starting node works, i.e. it does approximately balance the exploring procedure between Breadth-First Sampling and Depth-First Sampling methods. To learn the node embeddings, node2vec uses a decoder:

$$\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) = \frac{\exp^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{v_k \in \mathcal{V}} \exp^{\mathbf{z}_i^\top \mathbf{z}_j}} \qquad \approx p_{\mathcal{G},T(v_j|v_i)}, \tag{13}$$

where $p_{\mathcal{G},T(v_j|v_i)}$ is the probability of visiting v_j on a T-length random walk starting from v_i. Then the approach minimises:

$$\mathcal{L} = \sum_{(v_i,v_j)\in\mathcal{D}} -\log(\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j)), \tag{14}$$

where $\mathcal{D}$ is the random walks training set. For further information about the algorithm's optimisation, refer to [18]. To compare this work to our model, we use the node2vec version, which uses weights when doing the random walks. Furthermore, after obtaining the node embeddings, the link weight prediction is performed by:

$$e_{(v_i,v_j)} = z_i^\top z_j, \quad \text{with} \quad z_i, z_j \in Z = \text{node2vec}(G), \tag{15}$$

where G is the graph with the nodes, edges and weights: $G = (V, E, W)$.

Node2vec is an interesting, flexible and powerful method. It can be used in a wide variety of tasks and settings. Unlike our model, it is not explicitly designed for link weight prediction, so it is expected to perform worse. Random walk based approaches are dependent on the quality of the random walks, and those are highly dependent on the hyperparameter values. Thus, the necessity for finding the best possible hyperparameters for every dataset. Our model, however, is not that dependent on hyperparameters. Furthermore, LWP-WL always encodes structural roles and takes a precise edge-to-edge and node-to-edge filter when performing predictions. Node2vec, on the other hand, may not capture the structural role of every node in the graph necessarily and only takes into account the weights obtained in the random walk choosing process.

- Graph Convolutional Network (GCN) [19]: they learn node embeddings through graph convolution. For this, they use a message-passing framework where node embeddings are updated by aggregating embeddings of their neighbours. The layer-wise propagation rule for GCNs for the lth layer of a target v_i node to update is:

$$h_{v_i}^{(l+1)} = \sigma\left(\sum_j \frac{1}{c_{ij}} h_{v_j}^{(l)} W^{(l)}\right), \tag{16}$$

where v_j are the neighbouring nodes, h_v corresponds to the embedding of the node v, c_{ij} is a normalisation constant for the edge and $W^{(l)}$ is the matrix of weights for the lth layer. Then, we can use the output embeddings Z to perform and train any supervised task. For link weight prediction in this work, we follow the same framework as authors in [47] and calculate the link weight value for the edge (v_i, v_j) by:

$$e_{(v_i,v_j)} = z_i^\top z_j, \quad \text{with} \quad z_i, z_j \in Z = \mathrm{GCN}(X, A), \tag{17}$$

where X is the identity matrix of the graph (as we do not input features to the model) and A is the weighted adjacency matrix for the graph. Finally, we use the Mean Squared Error to optimise the model.

GCNs and LWP-WL share two main characteristics that make them comparable: (a) they both apply a structural role to the nodes by using the WL algorithm (see [19] Appendix A for the relation of GCNs with the WL algorithm), and (b) they both use convolutions to aggregate the nodes information. The main difference between LWP-WL and GCNs relies on how they perform the convolution. GCNs do a neighbourhood aggregation, while LWP-WL aggregate is based on the special node-to-node and node-to-edge filters presented in this work. In this manner, LWP-WL aggregates information in a more meaningful process for the link weight prediction task.

5.2. Setup

We perform the same experiment for each dataset. All the link weights are normalised using the L2 norm. Each experiment consists of 25 independent trials. In each trial, we randomly split the dataset into three subsets:

Table 2
Mean Squared Errors, standard deviation and improvement percentage for the compared approaches on all of the datasets, results for pWSBM, bWSBM, SBM, DCWBM and ModelR belong to [17].

	Airport	Collaboration	Congress	Forum
pWSBM	4.86e−2 ± 6e−4	4.07e−2 ± 1e−4	5.71e−2 ± 4e−4	7.26e−2 ± 3e−4
bWSBM	5.43e−2 ± 5e−4	4.62e−2 ± 1e−4	5.94e−2 ± 4e−4	8.45e−2 ± 3e−4
SBM	6.32e−2 ± 8e−4	4.97e−2 ± 3e−4	6.34e−2 ± 6e−4	8.51e−2 ± 4e−4
DCWBM	7.46e−2 ± 9e−4	5.00e−2 ± 2e−4	6.53e−2 ± 4e−4	8.82e−2 ± 4e−4
ModelR	1.3e−2 ± 1e−3	3.0e−2 ± 1e−3	3.6e−2 ± 3e−3	3.7e−2 ± 1e−3
Node2vec	3.73e−1 ± 1.89e−1	1e−3 ± 1e−3	6e−3 ± 7e−3	5.3e−2 ± 1.1e−2
GCN	2.75e−4 ± 3.7e−5	**1.20e−4 ± 5.8e−5**	**1.89e−4 ± 2.7e−5**	**3.6e−5 ± 5e−6**
LWP-WL	**1.88e−4 ± 4.6e−5**	1.39e−4 ± 1.36e−4	**1.95e−4 ± 3.4e−5**	7.2e−5 ± 1.56e−4

- 70% into the training set.
- 10% into validation set.
- 20% into the testing set.

The network contains a CNN in its first layer, which applies the edge to edge and node to edge filters. Then, four fully-connected layers of sizes 32, 16, 18, 1 are applied. All of the layers, but the last one use the ReLu activation. The size of the subgraph, K, is set to 10, and we perform gradient descent by the Adam algorithm with a learning rate of 0.001. We use the best possible hyperparameters provided by the authors for all compared models. We performed a bayesian hyperparameter optimisation for node2vec and GCNs since no link weight prediction experiments were reported in the original works. For node2vec with the following setting: learning rate in range $(0.001, 0.1)$, epochs in range $(100, 1000)$, random walk length in range $(30, 100)$, number of random walks in range $(8, 16)$, embedding size in range $(8, 20)$, p in range $(0.2, 1.0)$ and q in range $(1.0, 5.0)$. For GCN, we searched: learning rate and epochs in the same ranges as for node2vec, the hidden layer size in the range $(16, 64)$ and the embedding size in the range $(4, 32)$. The final values for node2vec were: epochs = 692, random walk length = 34, learning rate = 0.08, number of random walks = 11, p = 1.0, q = 1.24 and embedding size = 1. For GCN: epochs = 641, learning rate = 0.028 hidden layer size = 27 and embedding size = 26.

For our model, the parameters were selected based on a random hyperparameter search, with the first layer comprehending sizes from 64 to 18 and the consequent layers being half the size, the size of K was selected from $5, 8, 10, 15, 20, 30$ and the learning rate selected from $0.1, 0.01, 0.001$.

The size of the fully-connected layers is proportional to the complexity of the task, and the presented setting is standard in prediction tasks. The learning rate is also a common value for the Adam algorithm. Finally, the size of the subgraph and filters, K, was selected because its performance was the best overall. K is a hyperparameter that needs to be balanced between having enough information to properly perform the prediction and not having a big size where noisy and irrelevant nodes are added to the subgraph. We empirically obtained $K = 10$ to be the most suitable value for finding the most important nodes for the prediction and not adding too many nodes that worsen the performance of LWP-WL.

We use mean squared error (MSE) as the prediction accuracy metric. For each experiment, we report the mean and standard deviation of the error from all the trials.

The experiments ran on a Linux server with an i5-4590 CPU @ 3.30 GHz and TITAN X GPU with 12 GB of memory. All of the experiments used Adam optimiser with a batch size of 100.

5.3. Comparison

In this section, we compare the experimental results of LWP-WL with several techniques, including the state-of-the-art method, ModelR. We use the setup described in the previous subsection, the same used by the authors from ModelR and the modern methods node2vec and GCNs, which were never used for the link weight prediction task. The comparison is shown in Table 2.

Concerning Block Model-based approaches, we argue that our approach and ModelR can predict fine-grained per node weights while Block Models can only work at the Block level. This characteristic, and the non-linear learning ability, improves both deep learning-based approaches' performance and leave Block Models behind. Furthermore, we advocate that the Weighted-WL algorithm explicitly encodes Blocks into each node's label; however, we leave this study for a next work.

As results from the Table 2 show, our model outperforms the state-of-the-art method (ModelR). Unlike ModelR, our approach does encode structural roles due to the use of a node ordering algorithm (W-WL) based on Weisfeiler–Lehman. This feature allows our approach to better generalise over different graphs and subgraphs. Additionally, we use the adjacency matrices of extracted subgraphs as input to a Convolutional Neural Network (CNN). The use of this adjacency matrix improves our model's performance because CNNs can find patterns in links and weights between nodes, reinforced by the edge to edge and edge to nodes filters we developed for this task.

Node2vec presents good results on the task compared to block models and ModelR. However, the performance is lower when compared to GCNs and LWP-WL. Node2vec presents different performance between datasets (see Table 2). We hypothesise that the hyperparameter sensitivity is high for the model, and thus, each of the datasets requires further hyperparameter optimisation (we performed the hyperparameter search for the mean result in every dataset). Nevertheless, we choose to share the same hyperparameter setting for the comparison to be fair, in the same manner as we did for every model. Overall, node2vec is a model with good performance and scalability, but it lacks the characteristics that make GCNs and LWP-WL stand out, encoding structural roles and aggregating graph information.

GCNs prove themselves as a powerful method for the task. The results, as expected, are equivalent to LWP-WL, even outperforming on some of the datasets both in small-medium (see Table 2) and big sized graphs (see Table 3). The similarities that both methods share make them the best performing algorithms. They can both encode structural roles via the WL algorithm and apply convolution operations to aggregate information within the graph, being how they apply those convolutions the architectural difference between the methods. However, another aspect that differences LWP-WL from GCNs is scalability. Scalability is further analysed in the following subsection.

5.4. Scalability and complexity

We perform experiments with two big datasets (Geom and Astro) to address the scalability of our approach. We compare with ModelR since it is the state-of-the-art approach and GCNs due to their excellent performance and similarities with LWP-WL.

Table 3
Comparison between different methods and our approach, Mean Squared Errors, standard deviation, parameter amount and improvement percentage in the big-sized Astro and Geom datasets.

	Astro	Geom
ModelR	4.7e−2 ± 9e−3	2.17e−2 ± 2e−3
GCN	5.5e−5 ± 1.3e−5	**8e−6 ± 1e−6**
LWP-WL	**9e−6 ± 2e−6**	5.4e−5 ± 1.9e−5

As shown in Table 3, LWP-WL also outperforms ModelR for big datasets. Moreover, ModelR follows an embedding approach where each node is given a representation that is updated at training time. Authors of ModelR state that they set up the size of the embedding layer d following the rule $d = \log_2(N)$ where N is the number of nodes in the graph. Thus, the number of parameters in the embedding layer w_e scales (we ignore the following hidden layers since their size is not directly associated with the input) :

$$w_e = N \times d = N \log_2(N) \quad . \tag{18}$$

Thus, ModelR presents a highly scalable approach but incomparable performance-wise with LWP-WL.

On the other hand, GCNs scale in $\mathcal{O}(Lmd + Lmd^2)$ [48] being L the number of layers, m the number of edges and d the embedding size. This imposes a limitation on m the number of edges, which increases linearly and makes GCNs a less scalable model for big or growing graphs. Although some works proposed alternatives to overcome this limitation [49], it remains incomparable to LWP-WL as it only depends on the size of the subgraph K. Thus, although GCNs prove to be a reliable and well-performing model for the link weight prediction task, LWP-WL presents much better scalability making it suitable for larger graphs.

Unlike ModelR and GCNs, LWP-WL provides a user-defined parameter that constrains the number of parameters of the method, the size of the subgraph (K). This parameter defines the size of the subgraph to be the input to the neural network, keeping uniform the number of parameters needed by the LWP-WL model and doing it invariant to the number of nodes in the graph. The sum of the edge-to-edge and edge-to-node layers for the LWP-WL network will have parameters

$$w = K^2 C \tag{19}$$

where $C \in \mathbb{N}$ is the number of channels of the convolutional layers, we set it up to 10 by default.

However, the property of not scaling to the number of nodes in the graph comes at the cost of the computational and time complexity of the labelling algorithm. The W-WL algorithm takes K iterations to converge. The computations needed to complete the algorithm are the following, *(i)* the hash function for every node, performed in $\mathcal{O}(K)$ time, *(ii)* the evaluation of all hashes, it is done in $\mathcal{O}(K^2)$ time, and *(iii)* the sorting of the nodes which is completed in $\mathcal{O}(Klog(k))$ time. Therefore, the algorithm has $\mathcal{O}(K^2)$ time complexity in each iteration, and it needs $\mathcal{O}(K^3)$ time to finish in a graph of K nodes. It is important to notice that the labelling process is parallelizable.

We argue that in the tradeoff between having to complete the labelling algorithm in $\mathcal{O}(K^3)$ time for a subgraph of size K that usually varies between 5 and 25 (10 by default), and having a model that scales $N \log_2(N)$ for the full-size of the graph, LWP-WL is a more scalable approach that can handle big-sized graphs with even millions of nodes.

5.5. Ablation study

We also performed an ablation study with the final goal of understanding the impact level of different features of our model.

Table 4
Summary of the versions used for the ablation study.

Version	Node labelling	CNN
1	Random	No
2	W-WL	No
3	Random	Yes
4	W-WL	Yes

For this matter, we used a single dataset and measured the performance while gradually adding features. The purpose of this study is to gather reliable knowledge about our approach and the impact that different features have on it. There are two main features that we want to analyse.

Node labelling algorithm (W-WL). The W-WL node labelling algorithm for weighted graphs is one of the main contributions of this work. It provides consistency to the model by labelling roles that have a similar structural role with similar labels. We compare this algorithm with random labelling for nodes.

Convolutional Neural Network layer. The CNN is the first layer for the model that applies a unique edge to node and edge to edge filters to the input adjacency matrix. We want to measure this layer's impact in predicting and finding patterns between nodes, edges, and their weights.

We developed different versions of our approach by combining these features; a summary of those is shown in Table 4.

1. The baseline version, we do not add any feature to the approach. After extracting the enclosing subgraph, nodes are randomly ordered, and the flattened adjacency matrix is fed to a model without the CNN layer.
2. We add the W-WL node labelling algorithm to the baseline version.
3. In this version, CNN is used. However, nodes are randomly ordered.
4. Final version with all the features, the W-WL node labelling algorithm, the weighted adjacency matrix and the CNN layer are both used.

We used the Airport dataset for this study with the same setup used for the comparison subsection. The results obtained are shown in Table 5.

It is noticeable how features impact the model's performance, and the combination of these features allowed us to create a state-of-the-art approach for link weight prediction.

Results clearly demonstrate the effectiveness of the W-WL labelling algorithm, as it does significantly improve the versions that do not have it. Furthermore, the recent literature has demonstrated promising results on approaches using enclosing subgraph extraction. Besides our work, there have been two recent approaches for link prediction, WLNM [12], and SEAL [13], that achieved state-of-the-art results for the task. Therefore, a proper representation for the target link has been proven enough to predict links and their weights, avoiding whole processing graphs. However, it is compulsory to provide a node labelling technique to use an enclosing subgraph extraction approach. Further models can be generalised over different subgraphs by consistently labelling nodes to learn predictions. Additionally, the node labelling technique must preserve the topological directionality towards the target link to get the best performance, providing a mechanism for the model to focus on specific nodes.

Regarding the CNN layer, it does not improve the baseline when nodes are randomly ordered (version 1 to 3). However, it improves the efficiency when we apply the W-WL algorithm (version 3 to 4). This fact is due to the randomness of node order in the subgraph. When the labelling technique is applied, the nodes' structure and roles arise, providing proper criteria for the CNN to learn.

Table 5
Results for each version of the algorithm applied to the Airport dataset. We report the Pearson Correlation Coefficient and the Mean Squared errors with the Standard Deviation for 25 trials on each version. We also provide the improvement percentage of every version with regard to the previous one for each metric.

Ver.	MSE	Impr.	PCC	Impr.
1	1.89e−4 ± 2.6e−5		0.456 ± 0.033	
2	1.52e−4 ± 3.2e−5	24.34% (v1)	0.595 ± 0.129	23.36% (v1)
3	2.05e−4 ± 3.2e−5	−25.85% (v2)	0.412 ± 0.088	−44.41% (v2)
4	1.69e−4 ± 4.2e−5	21.3% (v3)	0.601 ± 0.127	31.44% (v3)

5.6. Experiments results and discussions

Finally, we present the conclusions and answers obtained to the questions formulated at the beginning of the section:

How well does LWP-WL perform in the link weight prediction task compared to the state-of-the-art models? Do the best performing models share any common characteristics?

The experiments performed in this section present the LWP-WL algorithm as the new state-of-the-art model for link weight prediction. The approach provides two main improvements over the compared models: **(a)** the extraction and ordering of nodes in subgraphs which provides a structural organisation pattern for the model, and **(b)** the performance and scalability obtained from the use of specific CNNs for the link weight prediction task.

Remarkably, the best performing models, i.e. LWP-WL, GCN, Node2vec and ModelR, are based on representation learning techniques. They all learn representations for the nodes in the graph to perform the link weight prediction task. Furthermore, LWP-WL, GCN and Node2vec use a convolution (Node2vec does it via random walk) to obtain such representations. We conclude that the models most capable of generating the best representations for the nodes in the graph are the most successful. To do so, aggregating information from the most significant neighbours and nodes is critical. However, the main issue is that models need consistent node ordering criteria when learning such representations. Otherwise, whether a node plays a vital role in the graph or lies as a simple disconnected node is irrelevant for the model. LWP-WL and GCNs are the best performing models because they encode structural roles via the WL algorithm, and thus, they have a criterion to generalise over training samples.

How scalable is LWP-WL compared to the best models? Does it perform well in big graphs? How can we achieve graph size-wise independent scalability?

We conclude that LWP-WL is a scalable approach that can handle big-sized graphs with even millions of nodes. Furthermore, the presented experiments show LWP-WL as an even contender with GCNs for big-sized graphs.

The main problem when working with graph-based methods is to make them independent of the graph size. The subgraph extraction method makes LWP-WL invariant to the number of nodes in the graph to solve such an issue, i.e. to predict a link weight, we only need a minor part of the graph. However, not scaling to the number of nodes in the graph comes at the cost of the computational and time complexity of the W-WL labelling algorithm. Nevertheless, the time needed to compute the W-WL algorithm for a given subgraph will always stay the same, while models that process the whole graph will linearly scale to the number of nodes in such graph.

What is the impact of the structural roles for nodes and, thus, the Weighted-WL graph labelling algorithm? And the special edge-to-edge and edge-to-node filters?

As we presented in the ablation study, the W-WL algorithm is the essential aspect of LWP-WL. The algorithm improves the model's performance, even making the LWP-WL without the CNN version significantly worse. This conclusion aligns with the strong performance of GCNs for the task. GCNs, as LWP-WL do also encode structural roles in the model, improving the performance and making GCNs a strong baseline.

Nevertheless, the CNN layer and the special edge-to-edge and edge-to-node filters prove themself an important addition to the model by improving its performance once the W-WL algorithm has been applied. Due to the structure that the algorithm provides, CNNs can properly apply convolutions over graph data and obtain superior performance compared to fully hidden layers.

6. Conclusion and future work

Inspired by the WLNM, we presented a link weight prediction approach, LWP-WL, that learns graph structure features from subgraphs. We have presented a graph labelling technique for weighted graphs that allows a neural network model to predict the weights of links. This neural network contains a Convolutional Neural Network layer that operates over the extracted subgraphs' adjacency matrices. It does apply unique edge to node and edge to edge filters taking advantage of the graph representation. Our experimental results present higher accuracy, performance and scalability than the state-of-the-art technique on every tested dataset. LWP-WL can learn graph structure features in different graphs, providing a solid baseline for an extensive set of applications.

We want to explore further the impact that Convolutional Neural Networks have on graph representations in the form of matrices. Finally, we also want to apply different strategies to the enclosing subgraph extraction process, focusing on strategies that might implicitly apply the Weisfeiler–Lehman algorithm.

CRediT authorship contribution statement

Unai Zulaika: Conceptualization, Methodology, Software, Investigation, Writing. **Rubén Sánchez-Corcuera:** Validation, Writing – review & editing. **Aitor Almeida:** Supervision, Writing – review & editing. **Diego López-de-Ipiña:** Project administration, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We gratefully acknowledge the support of the Basque Government's, Spain Department of Education for the predoctoral funding; the Ministry of Economy, Industry and Competitiveness of Spain under Grant No.: TIN2017-90042-R (SentientThings) and RTI2018-101045-A-C22 (FuturAAL). We would also like to thank NVIDIA for providing us their hardware via the Nvidia GPU Grant and the Basque Government for their Grant on scientific equipment adquisition with No. EC-19 .

References

[1] R.N. Lichtenwalter, J.T. Lussier, N.V. Chawla, New perspectives and methods in link prediction, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, pp. 243–252.
[2] P. Wang, B. Xu, Y. Wu, X. Zhou, Link prediction in social networks: the state-of-the-art, Sci. China Inf. Sci. 58 (1) (2015) 1–38.
[3] H. Chen, X. Li, Z. Huang, Link prediction approach to collaborative filtering, in: Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'05), IEEE, 2005, pp. 141–142.
[4] J. Lu, J. Xuan, G. Zhang, X. Luo, Structural property-aware multilayer network embedding for latent factor analysis, Pattern Recognit. 76 (2018) 228–241.
[5] Y. Lin, Z. Liu, M. Sun, Y. Liu, X. Zhu, Learning entity and relation embeddings for knowledge graph completion, in: Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
[6] X. Ma, P. Sun, G. Qin, Nonnegative matrix factorization algorithms for link prediction in temporal networks using graph communicability, Pattern Recognit. 71 (2017) 361–374.
[7] T. Oyetunde, M. Zhang, Y. Chen, Y. Tang, C. Lo, BoostGAPFILL: improving the fidelity of metabolic network reconstructions through integrated constraint and pattern-based methods, Bioinformatics 33 (4) (2016) 608–611.
[8] B. Geng, X. Zhou, Y. Hung, Growing enzyme gene networks by integration of gene expression, motif sequence, and metabolic information, Pattern Recognit. 42 (4) (2009) 557–561.
[9] E.M. Airoldi, D.M. Blei, S.E. Fienberg, E.P. Xing, Mixed membership stochastic blockmodels, J. Mach. Learn. Res. 9 (Sep) (2008) 1981–2014.
[10] R. Salakhutdinov, A. Mnih, Bayesian probabilistic matrix factorization using Markov chain Monte Carlo, in: Proceedings of the 25th International Conference on Machine Learning, ACM, 2008, pp. 880–887.
[11] D. Liben-Nowell, J. Kleinberg, The link-prediction problem for social networks, J. Am. Soc. Inf. Sci. Technol. 58 (7) (2007) 1019–1031.
[12] M. Zhang, Y. Chen, Weisfeiler-Lehman neural machine for link prediction, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2017, pp. 575–583.
[13] M. Zhang, Y. Chen, Link prediction based on graph neural networks, in: Advances in Neural Information Processing Systems, 2018, pp. 5171–5181.
[14] M.E. Newman, The structure and function of complex networks, SIAM Rev. 45 (2) (2003) 167–256.
[15] W. Li, X. Cai, Statistical analysis of airport network of China, Phys. Rev. E 69 (4) (2004) 046106.
[16] H. Jeong, S.P. Mason, A.-L. Barabási, Z.N. Oltvai, Lethality and centrality in protein networks, Nature 411 (6833) (2001) 41.
[17] Y. Hou, L.B. Holder, Deep learning approach to link weight prediction, in: 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 1855–1862.
[18] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 855–864.
[19] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2016, arXiv preprint arXiv:1609.02907.
[20] H.R. De Sá, R.B. Prudêncio, Supervised link prediction in weighted networks, in: The 2011 International Joint Conference on Neural Networks, IEEE, 2011, pp. 2281–2288.
[21] C. Fu, M. Zhao, L. Fan, X. Chen, J. Chen, Z. Wu, Y. Xia, Q. Xuan, Link weight prediction using supervised learning methods and its application to yelp layered network, IEEE Trans. Knowl. Data Eng. 30 (8) (2018) 1507–1518.
[22] S. Liu, X. Ji, C. Liu, Y. Bai, Similarity indices based on link weight assignment for link prediction of unweighted complex networks, Internat. J. Modern Phys. B 31 (2) (2017) 1650254.
[23] J. Zhao, L. Miao, J. Yang, H. Fang, Q.-M. Zhang, M. Nie, P. Holme, T. Zhou, Prediction of links and weights in networks by reliable routes, Sci. Rep. 5 (2015) 12261.
[24] E. Osaba, J. Del Ser, D. Camacho, M.N. Bilbao, X.-S. Yang, Community detection in networks using bio-inspired optimization: Latest developments, new results and perspectives with a selection of recent meta-heuristics, Appl. Soft Comput. 87 (2020) 106010.
[25] T. Murata, S. Moriyasu, Link prediction of social networks based on weighted proximity measures, in: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, IEEE Computer Society, 2007, pp. 85–88.
[26] X. Yang, B. Wang, Local ranking and global fusion for personalized recommendation, Appl. Soft Comput. 96 (2020) 106636.
[27] L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank citation ranking: Bringing order to the web, Tech. rep., Stanford InfoLab, 1999.
[28] N. Stanley, T. Bonacci, R. Kwitt, M. Niethammer, P.J. Mucha, Stochastic block models with multiple continuous attributes, Appl. Netw. Sci. 4 (1) (2019) 1–22.
[29] C. Aicher, A.Z. Jacobs, A. Clauset, Learning latent block structure in weighted networks, J. Complex Netw. 3 (2) (2014) 221–248.
[30] M. Duma, B. Twala, Optimising latent features using artificial immune system in collaborative filtering for recommender systems, Appl. Soft Comput. 71 (2018) 183–198.
[31] K. Miller, M.I. Jordan, T.L. Griffiths, Nonparametric latent feature models for link prediction, in: Advances in Neural Information Processing Systems, 2009, pp. 1276–1284.
[32] W.L. Hamilton, R. Ying, J. Leskovec, Representation learning on graphs: Methods and applications, 2017, arXiv preprint arXiv:1709.05584.
[33] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014. pp. 701–710.
[34] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications, AI Open 1 (2020) 57–81.
[35] M. Niepert, M. Ahmed, K. Kutzkov, Learning convolutional neural networks for graphs, in: International Conference on Machine Learning, 2016, pp. 2014–2023.
[36] T. Dettmers, P. Minervini, P. Stenetorp, S. Riedel, Convolutional 2d knowledge graph embeddings, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
[37] D.Q. Nguyen, T.D. Nguyen, D.Q. Nguyen, D. Phung, A novel embedding model for knowledge base completion based on convolutional neural network, 2017, arXiv preprint arXiv:1712.02121.
[38] B. Weisfeiler, A.A. Lehman, A reduction of a graph to a canonical form and an algebra arising during this reduction, Nauchno-Techn. Inform. 2 (9) (1968) 12–16.
[39] C.Y. Lee, An algorithm for path connections and its applications, IRE Trans. Electr. Comput. (3) (1961) 346–365.
[40] V. Colizza, R. Pastor-Satorras, A. Vespignani, Reaction–diffusion processes and metapopulation models in heterogeneous networks, Nat. Phys. 3 (4) (2007) 276.
[41] R.K. Pan, K. Kaski, S. Fortunato, World citation and collaboration networks: uncovering the role of geography in science, Sci. Rep. 2 (2012) 902.
[42] M.A. Porter, P.J. Mucha, M.E. Newman, C.M. Warmbrand, A network analysis of committees in the US house of representatives, Proc. Natl. Acad. Sci. 102 (20) (2005) 7057–7062.
[43] T. Opsahl, P. Panzarasa, Clustering in weighted networks, Social Networks 31 (2) (2009) 155–163.
[44] V. Batagelj, A. Mrvar, Pajek, in: Encyclopedia of Social Network Analysis and Mining, Springer, 2014, pp. 1245–1256.
[45] M.E. Newman, The structure of scientific collaboration networks, Proc. Natl. Acad. Sci. 98 (2) (2001) 404–409.
[46] P.W. Holland, K.B. Laskey, S. Leinhardt, Stochastic blockmodels: First steps, Social Networks 5 (2) (1983) 109–137.
[47] T.N. Kipf, M. Welling, Variational graph auto-encoders, 2016, arXiv preprint arXiv:1611.07308.
[48] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S.Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. (2020).
[49] W.L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, 2017, arXiv preprint arXiv:1706.02216.