Knowledge and Information Systems (2022) 64:743–771
https://doi.org/10.1007/s10115-021-01632-x

REGULAR PAPER

A hidden challenge of link prediction: which pairs to check?

Caleb Belth[1] · **Alican Büyükçakır**[1] · **Danai Koutra**[1]

Received: 10 March 2021 / Revised: 15 November 2021 / Accepted: 20 November 2021 /
Published online: 18 February 2022

Abstract
The traditional setup of link prediction in networks assumes that a test set of node pairs, which is usually balanced, is available over which to predict the presence of links. However, in practice, there is no test set: the ground truth is not known, so the number of possible pairs to predict over is quadratic in the number of nodes in the graph. Moreover, because graphs are sparse, most of these possible pairs will not be links. Thus, link prediction methods, which often rely on proximity-preserving embeddings or heuristic notions of node similarity, face a vast search space, with many pairs that are in close proximity, but that should not be linked. To mitigate this issue, we introduce LinkWaldo, a framework for choosing from this quadratic, massively skewed search space of node pairs, a concise set of candidate pairs that, in addition to being in close proximity, also *structurally resemble the observed edges*. This allows it to ignore some high-proximity but low-resemblance pairs, and also identify high-resemblance, lower-proximity pairs. Our framework is built on a model that theoretically combines stochastic block models (SBMs) with node proximity models. The block structure of the SBM maps out *where* in the search space new links are expected to fall, and the proximity identifies the most plausible links within these blocks, using locality sensitive hashing to avoid expensive exhaustive search. LinkWaldo can use any node representation learning or heuristic definition of proximity and can generate candidate pairs for any link prediction method, allowing the representation power of current and future methods to be realized for link prediction *in practice*. We evaluate LinkWaldo on 13 networks across multiple domains and show that on average it returns candidate sets containing 7–33% more missing and future links than both embedding-based and heuristic baselines' sets. Our code is available at https://github.com/GemsLab/LinkWaldo.

Keywords Link Prediction · Node similarity · Search space pruning · Representation learning · Locality Sensitive Hashing

✉ Caleb Belth
cbelth@umich.edu

Alican Büyükçakır
alicanb@umich.edu

Danai Koutra
dkoutra@umich.edu

[1] Computer Science & Engineering, University of Michigan, Ann Arbor, USA

1 Introduction

Link prediction is a long-studied problem that attempts to predict either missing links in an incomplete graph or links that are likely to form in the future. This has applications in discovering unknown protein interactions to speed up the discovery of new drugs, friend recommendation in social networks, knowledge graph completion, and more [1,20,21,32]. Techniques range from heuristics, such as predicting links based on the number of common neighbors between a pair of nodes, to machine learning techniques, which formulate the link prediction problem as a binary classification problem over node pairs [11,40].

Link prediction is often evaluated via a *ranking*, where pairs of nodes that are not currently linked are sorted based on the "likelihood" score given by the method being evaluated [21]. To construct the ranking, a "ground truth" test set of node pairs is constructed by either (1) removing a certain percentage of links from a graph at random or (2) removing the newest links that formed in the graph, if edges have timestamps. These removed edges form the test positives, and the same number of unlinked pairs is generated at random as test negatives. The methods are then evaluated on how well they are able to rank the test positives higher than the test negatives.

However, when link prediction is applied in practice, these ground truth labels are not known, since that is the very question that link prediction is attempting to answer. Instead, *any* pair of nodes that are not currently linked could link in the future. Thus, to identify likely missing or future links, a link prediction method would need to consider $O(n^2)$ node pairs for a graph with n nodes; most of which in sparse, real-world networks would turn out to not link. Proximity, on its own, is only a weak signal, sufficient to rank pairs in a balanced test set, but likely to turn up many false positives in an asymptotically skewed space, leaving discovering the relatively small number of missing or future links a challenging problem in practice, when no ground truth is known.

Proximity-based link prediction heuristics [20], such as Common Neighbors, could ignore some of the search space, such as nodes that are farther than two hops from each other, but this would not extend to other notions of proximity, like proximity-preserving embeddings. Duan et al. studied the problem of pruning the search space [8], but formulated it as top-k link prediction, which attempts to predict a small number of links, but misses a large number of missing links in the process, suffering from low recall.

The goal of this work is to develop a principled approach to choose, from the quadratic and skewed space of possible links, a set of candidate pairs for a link prediction method to make decisions about. We envision that this will allow current and future developments to be realized for link prediction in practice, where no ground truth set is available.

Problem 1 *Given a graph and a proximity function between nodes, we seek to return a candidate set of node pairs for a link predictor to make decisions about, such that the set is significantly smaller than the quadratic search space, but contains many of the missing and future links.*

Our insight to handle the vast number of negatives is to consider not *just* the proximity of nodes, but also their *structural resemblance* to observed links. We measure resemblance as the fraction of observed links that fall in inferred, graph-structural equivalence classes of node pairs. For example, Fig. 1 shows one possible grouping of nodes based on their degrees, where the resulting structural equivalence classes (the cells in the "roadmap") capture what fraction of observed links form between nodes of different degrees. Based on the roadmap, equivalence classes with a high fraction of observed edges are expected to contain more unlinked pairs than those with lower resemblance. We then employ node proximity within equivalence

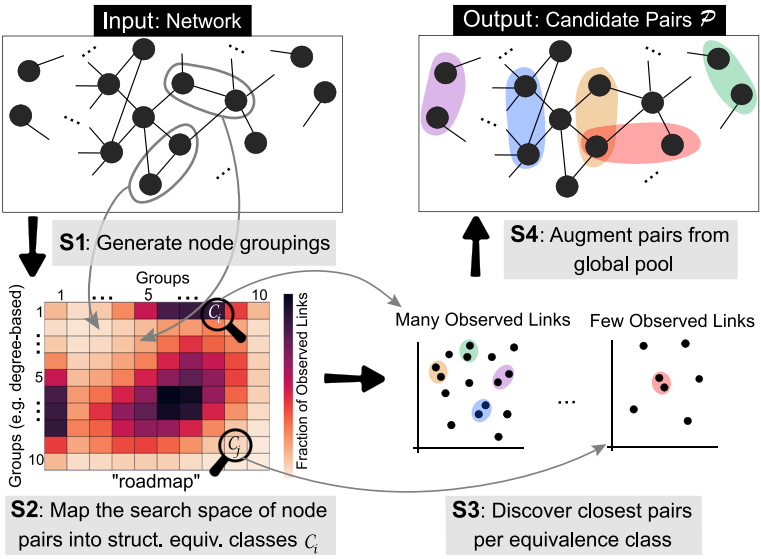


Fig. 1 Our proposed framework LINKWALDO chooses candidate pairs from the quadratic, highly skewed search space of possible links by first constructing a *roadmap*, which partitions the search space into structural equivalence classes of node pairs to capture how much pairs in each location resemble the *observed* links. This roadmap tells LINKWALDO how closely to look in each section of the search space. LINKWALDO follows the roadmap, selecting from each equivalence class the node pairs in closest proximity

classes, rather than globally, which decreases false positives that are in close proximity, but do not resemble observed links, and decreases false negatives that are farther away in the graph, but resemble many observed edges. Moreover, to avoid computing proximities for all pairs of nodes within each equivalence class, we extend self-tuning locality sensitive hashing (LSH). Our main **contributions** are:

- *Formulation & theoretical connections* Going beyond the heuristic of proximity between nodes, we model the plausibility of a node pair being linked as *both* their proximity *and* their structural resemblance to observed links. Based on this insight, we propose Future Link Location Models (FLLM), which combine proximity models and stochastic block models, and we prove that proximity models are a naive special case Sect. 3.
- *Scalable method* We develop a scalable method, LINKWALDO (Fig. 1), which implements FLLM, and uses locality sensitive hashing to implicitly ignore unimportant pairs Sect. 4.
- *Empirical analysis* We evaluate LINKWALDO on 13 diverse datasets from different domains, where it returns on average 22–33% more missing links than embedding-based models and 7–30% more than strong heuristics. We also investigate the effects of LINKWALDO's parameters Sect. 5.

Our code is at https://github.com/GemsLab/LinkWaldo.

2 Related work

In this paper, we focus on the understudied problem of choosing candidate pairs from the quadratic space of possible links, for link prediction methods to make predictions about. We first discuss link prediction techniques and methods that focus specifically on the problem

of selecting candidate pairs, and then briefly present work on node embeddings, since our proposed framework is flexible and can leverage latent representations to solve the problem at hand.

Link prediction Link prediction techniques range from heuristic definitions of similarity, such as Common Neighbors [20], Jaccard Similarity [20], and Adamic-Adar [1], to machine learning approaches, such as latent methods, which learn low-dimensional node representations that preserve graph-structural proximity in latent space [11], and GNN methods, which *learn* heuristics specific to each graph [40] or attempt to re-construct the observed adjacency matrix [15]. For detailed discussion of link prediction techniques, we refer readers to [20] and [21].

Selecting candidate pairs The closest problem to ours is top-k link prediction [8], which attempts to take a particular link prediction method and prune its search space to directly return the k highest score pairs. One method [8] samples multiple subgraphs to form a bagging ensemble, and performs NMF on each subgraph, returning the nodes with the largest latent factor products from each, while leveraging early stopping. The authors view their method's output as predictions rather than candidates, and thus focus on high precision at small values of k relative to our setting. Another approach, Approximate Resistance Distance Link Predictor [26] generates spectral node embeddings by constructing a low-rank approximation of the graph's *effective resistance* matrix, and applies a k-closest pairs algorithm on the embeddings, predicting these as links. However, this approach does not scale to moderate embedding dimensions (e.g., the dimensionality of 128 often-used used in embedding methods), and is often outperformed by the simple common neighbors heuristic.

A related problem is link recommendation, which seeks to identify the k most relevant nodes to a query node. It has been studied in social networks for friend recommendation [33], and in knowledge graphs [14] to pick subgraphs that are likely to contain links to a given query entity. In contrast, we focus on candidate pairs globally, not specific to a query node.

Node embeddings Node embeddings are low-dimensional latent representations of the nodes that preserve certain node properties. There are two main types of node embeddings [31,41]: proximity-preserving embeddings (e.g., [10,27,28,36]) capture a node's community structure (nodes that are close to each other are embedded similarly), whereas structural embeddings (e.g., [7,12,13,29]) capture roles and structural similarities between the nodes. Recent works have explored the connections and distinctions between proximity-preserving and structural embeddings [31,41].

Among the various embeddings that have been proposed in the recent years, we discuss a few methods that we leverage in our empirical evaluation. NetMF [28] is a proximity-preserving embedding that unifies skip-gram-based network embedding approaches (such as DeepWalk [27], LINE [36], PTE [35] and node2vec [10]) into closed-form matrix factorization. The difference between the embedding methods is determined by *what* matrix is factorized. Another approach, xNetMF [12], is also a matrix factorization-based node embedding approach, but it captures structural similarity rather than proximity-based similarity. Lastly, BINE [9] is specifically designed for embedding bipartite networks. It applies bias to its random walks according to nodes' centrality to preserve the long-tail distribution of the network, and its optimization function captures both first- and higher-order proximities among the nodes.

Usually structure-preserving embeddings are not used in link prediction, because structurally similar nodes are not necessarily likely to link (e.g., hub nodes are more likely to connect to spokes than to other hubs). However, our proposed framework is able to lever-

age information from both proximity-preserving and structural node embeddings: it can use representations (structural or proximity) in defining structural resemblance and then proximity-preserving embeddings in choosing pairs within structural equivalence classes. The details for these are discussed in Sect. 4.

3 Theory

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph or network with $|\mathcal{V}| = n$ nodes and $|\mathcal{E}| = m$ edges, where $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. The adjacency matrix $\mathbf{A}$ of $\mathcal{G}$ is an $n \times n$ binary matrix with element $a_{ij} = 1$ if nodes i and j are linked, and 0 otherwise. The set of node v's neighbors is $\mathcal{N}(v) = \{u : (u, v) \in \mathcal{E}\}$. We summarize the key symbols used in this paper and their descriptions in Table 1.

We now formalize the problem that we seek to solve:

Problem 2 *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a proximity function $sim : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^+$ between nodes, and a budget $k << n^2$, return a set of plausible candidate node pairs $\mathcal{P} \subset \mathcal{V} \times \mathcal{V}$ of size $|\mathcal{P}| = k$ for a link predictor to make decisions about.*

We describe next how to define resemblance in a principled way inspired by stochastic block models, introduce a unified model for link prediction methods that use the proximity of nodes to rank pairs, and describe our model, which combines resemblance and proximity to solve Problem 2.

3.1 Stochastic block models

Stochastic block models (SBMs) are generative models of networks. They model the connectivity of graphs as emerging from the community or group membership of nodes [25].

Node Grouping. A node grouping Γ is a set of groups or subsets $\mathcal{V}_i$ of the nodes that satisfies $\bigcup_{\mathcal{V}_i \in \Gamma} \mathcal{V}_i = \mathcal{V}$. It is called a *partition* if it also satisfies $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset \; \forall \; \mathcal{V}_i \neq \mathcal{V}_j \in \Gamma$. Each node $v \in \mathcal{V}$ has a $|\Gamma|$-dimensional binary membership vector $\boldsymbol{\mu}_v$, with element $\mu_{vi} = 1$ if v belongs to group $\mathcal{V}_i$.

Table 1 Description of major symbols used throughout the paper

Notation	Description
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Graph, nodes, edges
$\lvert\mathcal{V}\rvert = n, \lvert\mathcal{E}\rvert = m$	Number of nodes resp. edges in $\mathcal{G}$
$\mathcal{E}_{\text{new}}$	Unobserved future or missing links
Γ, Π	Grouping of $\mathcal{V}$, Partition of $\mathcal{V} \times \mathcal{V}$
$\mathbf{A}$	Adjacency matrix
$\mathbf{X} \in \mathbb{R}^{n \times d}$	Node embedding matrix
$\mathbf{x}_v \in \mathbb{R}^d$	Embedding vector of node v
$\boldsymbol{\mu}_v$	Membership vector of node v
$\mathcal{C}_i$	Equivalence class i
$\mathcal{P}$, $\tilde{\mathcal{P}}_G$	Pairs selected by LinkWaldo, global pool
k, κ	Budget for $\lvert\mathcal{P}\rvert$, target for an equivalence class

A node grouping can capture community structure, but it can also capture other graph-structural properties, like the degrees of nodes, in which case the SBM captures the compatibility of nodes with respect to degree—namely, degree assortativity [24].

Membership Indices. The membership indices $I_{u,v}$ of nodes u, v are the set of group ids (i, j) such that $u \in \mathcal{V}_i$ and $v \in \mathcal{V}_j$: $I_{u,v} \triangleq \{i : \mu_{u,i} = 1\} \times \{j : \mu_{v,j} = 1\},\ i, j \in \{1, 2, \ldots, |\Gamma|\}$.

Membership equivalence relation & classes. The membership indices form the equivalence relation $\sim_I$ where $(u, v) \sim_I (u', v') \iff I_{u,v} = I_{u',v'}$. This induces a partition $\Pi = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{|\Pi|}\}$ over all pairs of nodes $\mathcal{V} \times \mathcal{V}$ (both linked and unlinked), where the equivalence class $\mathcal{C}_i$ contains all node pairs (u, v) with the same membership indices, i.e., $\boldsymbol{\mu}_u = \boldsymbol{\mu}$ and $\boldsymbol{\mu}_v = \boldsymbol{\mu}'$ for some $\boldsymbol{\mu}, \boldsymbol{\mu}' \in \{0, 1\}^{|\Gamma|}$. We denote the equivalence class of pair (u, v) as $[(u, v)]_{\sim_I}$.

Example 1 If nodes are grouped by their degrees to form Γ, then the membership indices $I_{u,v}$ of node pair (u, v) are determined by u and v's respective degrees. For example, in Fig. 1, the upper circled node pair has degrees 3 and 5, respectively, which determines their equivalence class—in this case, the cell (3, 5) in the roadmap. Each cell of the roadmap corresponds to an equivalence class $\mathcal{C}_i \in \Pi$.

We can now formally define an SBM:

Definition 1 (*Stochastic block model—SBM*) Given a node grouping Γ and a $|\Gamma| \times |\Gamma|$ weight matrix $\mathbf{W}$ specifying the propensity for links to form across groups, the probability that two nodes link given their group memberships is $\Pr(a_{uv} = 1 | \boldsymbol{\mu}_u, \boldsymbol{\mu}_v) = \sigma(\boldsymbol{\mu}_u^T \mathbf{W} \boldsymbol{\mu}_v)$, where function $\sigma(\cdot)$ converts the dot product to a probability (e.g., sigmoid) [23].

The vanilla SBM [25] assigns each node to one group (i.e., the grouping is a partition and membership vectors $\boldsymbol{\mu}$ are one-hot), in which case $\boldsymbol{\mu}_u^T \mathbf{W} \boldsymbol{\mu}_v = w_{I_{u,v}}$. The overlapping SBM [17,23] is a generalization that allows nodes to belong to multiple groups, in which case membership vectors may have multiple elements set to 1, and $\boldsymbol{\mu}_u^T \mathbf{W} \boldsymbol{\mu}_v = \sum_{i,j \in I_{u,v}} w_{ij}$.

Resemblance. Given an SBM with grouping Γ, we define the resemblance of node pair $(u, v) \in \mathcal{V} \times \mathcal{V}$ under the SBM as the percentage of the observed (training) edges that have the same group membership as (u, v):

$$\rho(u, v) \triangleq \frac{|\{(v_1, v_2) \in \mathcal{E} : (v_1, v_2) \sim_I (u, v)\}|}{m}. \tag{1}$$

Example 2 In Fig. 1, the resemblance $\rho(u, v)$ of node pair (u, v) corresponds to the density of the cell that it maps to. The high density in the border cells indicates that many low-degree nodes connect to high-degree nodes. The dense central cells indicate that mid-degree nodes connect to each other.

3.2 Proximity models

Proximity-based link prediction models (PM) model the connectivity of graphs based on the proximity of nodes. Some methods define the proximity of nodes with a sensible heuristic, such as Common Neighbors (CN), Jaccard Similarity (JS), and Adamic/Adar (AA). More recent approaches learn latent similarities between nodes, capturing the proximity in latent

embeddings such that nodes that are in close proximity in the graph have similar latent embeddings, often using dot product as the measure of latent similarity [11].

Node Embedding. A node embedding, $\mathbf{x}_v \in \mathbb{R}^d$, is a real-valued, d-dimensional vector representation of a node $v \in \mathcal{V}$. We denote all the node embeddings as a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$.

Definition 2 (*Proximity model—PM*) Given a similarity or proximity function $sim : \mathcal{V} \times \mathcal{V} \to \mathbb{R}^+$ between nodes, the probability that nodes u and v link is an increasing function of their proximity: $\Pr(a_{uv} = 1 | sim(\cdot, \cdot)) = f(sim(u, v))$.

Instances of the PM include the latent proximity model:

$$sim_{\text{LaPM}}(u, v) \triangleq \mathbf{x}_u^T \mathbf{x}_v, \tag{2}$$

where $\mathbf{x}_u, \mathbf{x}_v$ are the nodes' latent embeddings; and the Common Neighbors, Jaccard Similarity, and Adamic/Adar models:

$$sim_{\text{CN}}(u, v) \triangleq |\mathcal{N}(u) \cap \mathcal{N}(v)|, \tag{3}$$

$$sim_{\text{JS}}(u, v) \triangleq \frac{|\mathcal{N}(u) \cap \mathcal{N}(v)|}{|\mathcal{N}(u) \cup \mathcal{N}(v)|}, \text{ and} \tag{4}$$

$$sim_{\text{AA}}(u, v) \triangleq \sum_{v' \in \mathcal{N}(u) \cap \mathcal{N}(v)} \frac{1}{\log |\mathcal{N}(v')|}. \tag{5}$$

3.3 Proposed: future link location model

Unlike SBM and PM, our model, which we call the *Future Link Location Model* (FLLM), is not just modeling the probability of links, but rather *where* in the search space future links are likely to fall. To do so, FLLM uses a partition of the search space, and corresponding SBM, as a roadmap that gives the number of new edges expected to fall in each equivalence class. To formalize this idea, we first define two distributions:
New and Observed Distributions. The new link distribution $p_{\text{n}}(\mathcal{C}_i) \triangleq \Pr(\mathcal{C}_i | \mathcal{E}_{\text{new}})$ and the observed link distribution $p_{\text{o}}(\mathcal{C}_i) \triangleq \Pr(\mathcal{C}_i | \mathcal{E})$ capture the fraction of new and observed edges that fall in equivalence class $\mathcal{C}_i$, respectively. The new link distribution is unobserved.

Definition 3 (*Future link location model—FLLM*) Given an overlapping SBM with grouping Γ, the expected number of new links in equivalence class $\mathcal{C}_i$ is proportional to the number of *observed* links in $\mathcal{C}_i$, and the probability of node pair (u, v) linking is equal to the pair's resemblance times their proximity relative to other nodes in $[(u, v)]_{\sim_I}$:

$$\Pr(a_{uv} = 1 | \boldsymbol{\mu}_u, \boldsymbol{\mu}_v, sim(\cdot, \cdot)) = \rho(u, v) \cdot \frac{sim(u, v)}{\sum_{(u', v') \in [(u, v)]_{\sim_I}} sim(u', v')} .$$

FLLM depends on the following theorem, which states that if $q\%$ of the observed links fall in equivalence class $\mathcal{C}_i$, then in expectation, $q\%$ of the *unobserved* links will fall in equivalence class $\mathcal{C}_i$. We initially assume that the unobserved future links follow the same distribution as the observed links—as generally assumed in machine learning—that is, the relative fraction of links in each equivalence class will be the same for future links as observed links: $p_{\text{n}} = p_{\text{o}}$. In the next subsection, we show that for a fixed k, the error in this assumption is determined by the total variation distance between p_{n} and p_{o}, and hence is upper bounded by a constant.

Theorem 1 *Given an overlapping SBM with grouping* Γ *inducing the partition* Π *of* $\mathcal{V} \times \mathcal{V}$ *for a graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$*, out of k new (unobserved) links* $\mathcal{E}_{new}$*, the expected number that will fall in equivalence class* $\mathcal{C}_i$ *and its variance are:*

$$\mathbb{E}[|\mathcal{C}_i \cap \mathcal{E}_{new}|] = \frac{k|\mathcal{C}_i \cap \mathcal{E}|}{m} \tag{6}$$

$$Var(|\mathcal{C}_i \cap \mathcal{E}_{new}|) = \frac{k|\mathcal{C}_i \cap \mathcal{E}||\mathcal{E} \setminus \mathcal{C}_i|}{m^2}. \tag{7}$$

Proof Each of the k new edges either falls in equivalence class $\mathcal{C}_i$ or it does not, and the probability of the former outcome is defined above as $\Pr(\mathcal{C}_i|\mathcal{E}_{\text{new}})$. Thus, the number of the k new edges that fall in equivalence class $\mathcal{C}_i$—i.e., $|\mathcal{C}_i \cap \mathcal{E}_{\text{new}}|$—is a binomial random variable over k trials, with success probability $\Pr(\mathcal{C}_i|\mathcal{E}_{\text{new}})$. This binomial random variable has expected value

$$\mathbb{E}[|\mathcal{C}_i \cap \mathcal{E}_{\text{new}}|] = k\Pr(\mathcal{C}_i|\mathcal{E}_{\text{new}}), \tag{8}$$

and variance

$$Var(|\mathcal{C}_i \cap \mathcal{E}_{\text{new}}|) = k\Pr(\mathcal{C}_i|\mathcal{E}_{\text{new}})(1 - \Pr(\mathcal{C}_i|\mathcal{E}_{\text{new}})). \tag{9}$$

We can derive $\Pr(\mathcal{C}_i|\mathcal{E}_{\text{new}})$ via $\Pr(\mathcal{C}_i|\mathcal{E})$ and Bayes' rule:

$$\begin{aligned}
\Pr(\mathcal{C}_i|\mathcal{E}) &= \frac{\Pr(\mathcal{E}|\mathcal{C}_i)\Pr(\mathcal{C}_i)}{\Pr(\mathcal{E})} && \text{(Bayes' rule)} \\
&= \frac{\frac{|\mathcal{C}_i \cap \mathcal{E}|}{|\mathcal{C}_i|}\frac{|\mathcal{C}_i|}{|\mathcal{V}\times\mathcal{V}|}}{m/|\mathcal{V} \times \mathcal{V}|} && \text{(Plugging in definitions)} \\
&= \frac{|\mathcal{C}_i \cap \mathcal{E}|}{m}. && \text{(Simplifying)}
\end{aligned}$$

Combining the last equation with Eq. (8) results directly in Eq. (6), and by substituting into Eq. (9) we obtain:

$$Var(|\mathcal{C}_i \cap \mathcal{E}_{\text{new}}|) = \frac{k|\mathcal{C}_i \cap \mathcal{E}|}{m}(1 - \frac{|\mathcal{C}_i \cap \mathcal{E}|}{m}) = \frac{k|\mathcal{C}_i \cap \mathcal{E}||\mathcal{E} \setminus \mathcal{C}_i|}{m^2},$$

where we used the fact that $|\mathcal{E}\setminus\mathcal{C}_i| = |\mathcal{E}| - |\mathcal{C}_i \cap \mathcal{E}|$. □

3.4 Guarantees on error

There are three possible sources of error in the FLLM. First, the budget k could be set too small to return all the missing links (Typ1- Err). Second, the budget could be distributed inappropriately across the equivalence classes, which only happens if the future link distribution differs from the observed link distribution (Typ2- Err). Finally, wrong pairs could be returned from some equivalence classes (Typ3- Err); for instance, if one equivalence class is assigned 10 pairs and there are in fact 10 unlinked pairs in it, some of the 10 returned may be different from the 10 unlinked pairs, resulting in error.

Typ1- Err is not inherent to FLLM: it stems from resource constraints. Since the returned pairs $\mathcal{P}$ are candidate pairs for a link predictor to make decisions about, the only limiting factor on k is how many decisions the link predictor can feasibly make. We discuss this further in a Connectomics application in Sect. 6. Furthermore, Typ3- Err arises from the method

used to return pairs from equivalence classes, rather than from FLLM. We discuss means of alleviating this error in method Sect. 4.

Only TYP2- ERR is inherent to FLLM. However, we now show that this error is upper bounded by a constant. While the derivation in the previous subsection assumed that the future link distribution is the same as the observed link distribution, we show that for a fixed k, the amount of error incurred when this assumption does not hold is entirely dependent on the *total variation distance* and hence is upper bounded by $2k$.

Total Variation Distance. The total variation distance [37] between p_n and p_o, which is a metric, is defined as

$$d_{TV}(p_\mathrm{n}, p_\mathrm{o}) \triangleq \sup_{\mathcal{A}\subset\Pi} |p_\mathrm{n}(\mathcal{A}) - p_\mathrm{o}(\mathcal{A})|. \tag{10}$$

Total Error. The total error made in the approximation of $\mathbb{E}[|\mathcal{C}_i \cap \mathcal{E}_\mathrm{new}|]$ using Eq. (6) is defined as

$$\begin{aligned}\xi &\triangleq \sum_{\mathcal{C}_i\in\Pi} |\hat{\mathbb{E}}[|\mathcal{C}_i \cap \mathcal{E}_\mathrm{new}|] - \mathbb{E}[|\mathcal{C}_i \cap \mathcal{E}_\mathrm{new}|]| \\ &= \sum_{\mathcal{C}_i\in\Pi} |kp_\mathrm{o}(\mathcal{C}_i) - kp_\mathrm{n}(\mathcal{C}_i)|, \end{aligned} \tag{11}$$

where $\hat{\mathbb{E}}[|\mathcal{C}_i \cap \mathcal{E}_\mathrm{new}|]$ is the true expected value regardless of whether or not $p_\mathrm{n} = p_\mathrm{o}$ holds.

Theorem 2 *The total error incurred over Π in the computation of the expected number of new edges that fall in each $\mathcal{C}_i \in \Pi$ is an increasing function of the number of new pairs k and the total variation distance between p_n and p_o. Furthermore, it has the following upper bound:*

$$\xi = 2k\, d_{TV}(p_n, p_o) \leq \min(2k, 2k\sqrt{1/2 D_{KL}(p_n||p_o)}). \tag{12}$$

Proof

$$\begin{aligned}\xi &\triangleq \sum_{\mathcal{C}_i\in\Pi} |kp_\mathrm{o}(\mathcal{C}_i) - kp_\mathrm{n}(\mathcal{C}_i)| && \text{See Eq. (11)} \\ &= 2kd_{TV}(p_\mathrm{n}, p_\mathrm{o}) && \text{See chapter 5 of [19]} \\ &\leq \min(2k, 2k\sqrt{1/2 D_\mathrm{KL}(p_\mathrm{n}||p_\mathrm{o})}) \end{aligned}$$

The inequality on the last line holds based on the fact that (1) $d_{TV}(\cdot,\cdot)$ ranges in $[0, 1]$ and thus $d_{TV}(p_\mathrm{n}, p_\mathrm{o}) \leq 1$, and (2) Pinsker's inequality [37], which upper bounds $d_{TV}(\cdot,\cdot)$ via KL-divergence: $d_{TV}(p_\mathrm{n}, p_\mathrm{o}) \leq \sqrt{1/2 D_\mathrm{KL}(p_\mathrm{n}||p_\mathrm{o})}$. □

3.5 Proximity model as a special case of FLLM

The PM, defined in Sect. 3.2, is a special case of FLLM, where FLLM's grouping contains just one group $\Gamma = \{\mathcal{V}\}$. That is, if the nodes are not grouped, then the models give the same result. Thus, FLLM's improvement over LaPM is a result of using structurally meaningful groupings over the graph. The following theorem states this result formally.

Theorem 3 *For a single-node grouping $\Gamma = \{\mathcal{V}\}$, both PM and FLLM give the same ranking of pairs $(u, v) \in \mathcal{V} \times \mathcal{V}$:*

$$Pr_{PM}(a_{uv} = 1|sim(\cdot,\cdot)) > Pr_{PM}(a_{u'v'} = 1|sim(\cdot,\cdot)) \iff$$

$$Pr_{FLLM}(a_{uv} = 1|\boldsymbol{\mu}_u, \boldsymbol{\mu}_v, sim(\cdot,\cdot)) > Pr_{FLLM}(a_{u'v'} = 1|\boldsymbol{\mu}_{u'}, \boldsymbol{\mu}_{v'}, sim(\cdot,\cdot)).$$

Proof Since $\Gamma = \{\mathcal{V}\}$ the partition contains just one equivalence class $\Pi = \{\mathcal{V} \times \mathcal{V}\}$, and since $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, all observed edges fall in this lone equivalence class $\mathcal{C} = \mathcal{V} \times \mathcal{V}$. Thus $\rho(u, v) = 1\ \forall (u, v) \in \mathcal{V} \times \mathcal{V}$.

Since there is only one equivalence class, the denominator in Definition 3 is equal to a constant $c \triangleq \sum_{(u',v') \in [(u,v)]_{\sim_I}} sim(u', v') = \sum_{(u',v') \in \mathcal{V} \times \mathcal{V}} sim(u', v')\ \forall (u, v) \in \mathcal{V} \times \mathcal{V}$. Therefore, $\text{Pr}_{\text{FLLM}}(a_{uv} = 1|\boldsymbol{\mu}_u, \boldsymbol{\mu}_v, sim(\cdot,\cdot)) = \frac{1}{c} sim(u, v)$, and both models are increasing functions of $sim(\cdot,\cdot)$. □

4 Method

We solve Problem 2 by using our FLLM model in a new method, LINKWALDO, shown in Fig. 1, which has four steps:

- **S1**: Generate node groupings and equivalence classes.
- **S2**: Map the search space, deciding how many candidate pairs to return from each equivalence class.
- **S3**: Search each equivalence class, returning directly the highest proximity pairs, and stashing some slightly lower-proximity pairs in a global pool.
- **S4**: Choose the best pairs from the global pool to augment those returned from each equivalence class.

We discuss these steps next, give pseudocode in Algorithm 1, and discuss time complexity at the end of the section.

4.1 (S1) Generating node groupings

In theory, we would like to infer the groupings that directly maximize the likelihood of the observed adjacency matrix. However, the techniques for inferring these groupings (and the corresponding node membership vectors) are computationally intensive, relying on Markov chain Monte Carlo (MCMC) methods [22]. Indeed, these methods are generally applied in networks with only up to a few hundred nodes [23]. In cases where n is large enough that considering all $O(n^2)$ node pairs would be computationally infeasible, so would be MCMC. Instead LINKWALDO uses a fixed grouping, though it is agnostic to how the nodes are grouped. We discuss a number of sensible groupings below, and discuss how to set the number of groups in Sect. 5.3. Any other grouping can be readily used within our framework, but should be carefully chosen to lead to strong results.

- *Log-binned Node Degree (DG).* This grouping captures degree assortativity [24]—the extent to which low degree nodes link with other low degree nodes vs. high degree nodes—by creating uniform bins in log-space (Fig. 1 gives an example with linear bins).
- *Structural Embedding Clusters (SG).* This grouping extends DG by clustering latent node embeddings that capture structural roles of nodes [31].
- *Communities (CG).* This grouping captures community structure by clustering proximity preserving latent embeddings or using community detection methods.
- *Multiple Groupings (MG).* Any subset of these groupings or any other groupings can be combined into a new grouping, by setting $\boldsymbol{\mu}_v$ element(s) to 1 for v's membership in each grouping, since nodes can have overlapping group memberships.

Algorithm 1 LINKWALDO($\mathcal{G}, sim(\cdot,\cdot), k, \tau$)

/* **S1**: Generating Node Groupings */
Generate node grouping Γ inducing partition Π ▷ § 4.1
$\mathcal{P}, \tilde{\mathcal{P}}_G \leftarrow \emptyset, \emptyset$ ▷ Initialize pairs to return and global pool
for $\mathcal{C}_i \in \Pi$ **do** ▷ Search each equivalence class § 4.3
  /* **S2**: Mapping the Search Space */
  $\bar{\mu} \leftarrow \mathbb{E}[|\mathcal{C}_i \cap \mathcal{E}_{\text{new}}|]$ ▷ Eq. (6)
  $\sigma \leftarrow \sqrt{Var(|\mathcal{C}_i \cap \mathcal{E}_{\text{new}}|)}$ ▷ Eq. (7)
  /* **S3**: Discovering Closest Pairs per Equivalence Class */
  if $|\mathcal{C}_i| < \tau$ **then**
    SELECTPAIRSEXACT($\mathcal{P}, \tilde{\mathcal{P}}_G, \mathcal{C}_i, \bar{\mu}, \sigma$)
  else
    SELECTPAIRSAPPROX($\mathcal{P}, \tilde{\mathcal{P}}_G, \mathcal{C}_i, \bar{\mu}, \sigma$)
/* **S4**: Augmenting Pairs from Global Pool */
$\mathcal{P} \leftarrow \mathcal{P} \cup \{\text{top } k - |\mathcal{P}| \text{ pairs from } \tilde{\mathcal{P}}_G\}$
return $\mathcal{P}$

procedure SELECTPAIRSEXACT($\mathcal{P}, \tilde{\mathcal{P}}_G, \mathcal{C}_i, \bar{\mu}, \sigma$)
  Sort pairs $(u, v) \in \mathcal{C}_i$ in descending order on $sim(u, v)$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\text{top } \bar{\mu} - \sigma \text{ pairs}\}$
  $\tilde{\mathcal{P}}_G \leftarrow \tilde{\mathcal{P}}_G \cup \{\text{next } 2\sigma \text{ pairs}\}$

procedure SELECTPAIRSAPPROX($\mathcal{P}, \tilde{\mathcal{P}}_G, \mathcal{C}_i, \bar{\mu}, \sigma$)
  for $i = 1, 2, \ldots, r$ **do** ▷ Create r trees
    $\mathcal{B} \leftarrow \{(\mathcal{V}_u, \mathcal{V}_v)\}$ ▷ buckets start off as the root
    while $\text{Vol}(\mathcal{B}) > \kappa$ **do** ▷ cf. Prob. 3 for κ definition
      Choose $h(\cdot)$ at random from $\mathcal{H}_{\text{rh}}$
      $\mathcal{B}' \leftarrow \emptyset$ ▷ Create new buckets
      for $\beta \in \mathcal{B}$ **do** ▷ Branch each leaf (bucket)
        $\beta'_{\text{left}} \leftarrow (\{u \in \mathcal{V}_u^{(\beta)} : h(\mathbf{x}_u) < 0\}, \{v \in \mathcal{V}_v^{(\beta)} : h(\mathbf{x}_v) < 0\})$
        $\beta'_{\text{right}} \leftarrow (\{u \in \mathcal{V}_u^{(\beta)} : h(\mathbf{x}_u) \geq 0\}, \{v \in \mathcal{V}_v^{(\beta)} : h(\mathbf{x}_v) \geq 0\})$
        $\mathcal{B}' \leftarrow \mathcal{B}' \cup \{\beta_{\text{left}}, \beta_{\text{right}}\}$
      if $\text{Vol}(\mathcal{B}) \leq \kappa$ **then** $\mathcal{B} \leftarrow \mathcal{B}'$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\text{top } \bar{\mu} - \sigma \text{ pairs}\}$
  $\tilde{\mathcal{P}}_G \leftarrow \tilde{\mathcal{P}}_G \cup \{\text{next } 2\sigma \text{ pairs}\}$

4.2 (S2) Mapping the search space

LINKWALDO's approach to mapping the search space (i.e., identifying how many pairs to return per equivalence class $\mathcal{C}_i$) follows directly from Theorem 1. LINKWALDO computes the expected number of pairs in each equivalence class based on Eq. (6) and its variance based on Eq. (7), as a measure of the uncertainty. When LINKWALDO searches each equivalence class $\mathcal{C}_i$, it returns the expected number of pairs *minus* a standard deviation *directly* and adds more pairs, up to a standard deviation past the mean, to a global pool $\tilde{\mathcal{P}}_G$, for use in **S4**. Thus, LINKWALDO adds into $\mathcal{P}$ the $\mathbb{E}[|\mathcal{C}_i \cap \mathcal{E}_{new}|] - \sqrt{Var(|\mathcal{C}_i \cap \mathcal{E}_{\text{new}}|)}$ pairs in closest proximity in equivalence class $\mathcal{C}_i$, and the next $2\sqrt{Var(|\mathcal{C}_i \cap \mathcal{E}_{\text{new}}|)}$ closest pairs into the global pool $\tilde{\mathcal{P}}_G$ (both expressions are rounded to the nearest integer). Node pairs that are already linked are skipped.

4.3 (S3) discovering closest pairs per equivalence class

We now discuss how LinkWaldo discovers the κ closest unlinked pairs *within* each equivalence class (Fig. 1), where κ is determined in step **S2** based on the expected number of pairs in the equivalence class, and variance (uncertainty).

Problem 3 *Given an equivalence class $\mathcal{C}_i$, return the top-κ unlinked pairs in $\mathcal{C}_i$ in closest proximity $sim(\cdot,\cdot)$, where $\kappa = \mathbb{E}[|\mathcal{C}_i \cap \mathcal{E}_{new}|] + \sqrt{Var(|\mathcal{C}_i \cap \mathcal{E}_{new}|)}$ (based on **S2**).*

For equivalence classes smaller than some tolerance τ, it is feasible to search all pairs of nodes exhaustively. However, for $|\mathcal{C}_i| > \tau$, this should be avoided, to make the search practical. We first discuss this case when using the dot product similarity $sim_{\text{LaPM}}(\cdot,\cdot)$ in Eq. (2) and then discuss it for other similarity models (CN, JS, and AA) given by Eqs. (3)–(5). Finally, we introduce a refinement that improves the robustness of LinkWaldo against errors in proximity.

4.3.1 Avoiding exhaustive search for dot product

In the case of dot product, we use Locality Sensitive Hashing (LSH) [39] to avoid searching all $|\mathcal{C}_i|$ pairs. LSH functions have the property that the probability of two items colliding is a function of their similarity. We use the following fact, which is a direct consequence of the definition of the membership equivalence relation given in Sect. 3:

Fact 1 *The equivalence class $\mathcal{C}_i$ can be decomposed into the Cartesian product of two sets $\mathcal{C}_i = \mathcal{V}_u \times \mathcal{V}_v$, where $\mathcal{V}_u \triangleq \{u : \boldsymbol{\mu}_u = \boldsymbol{\mu}\}$ and $\mathcal{V}_v = \{v : \boldsymbol{\mu}_v = \boldsymbol{\mu}'\}$.*

At a high level, to solve Problem 3, we hash each node embedding of the nodes in $\mathcal{V}_u$ and $\mathcal{V}_v$ using a locality sensitive hash function. We design the hash function, described next, such that the number of pairs that map to the same bucket is greater than κ, but as small as possible, to maximally prune pairs. Once the embeddings are hashed, we search the pairs in each hash bucket for the κ closest. We normalize the embeddings so that dot product is equivalent to cosine similarity, and use the Random Hyperplane LSH family [6].

Definition 4 (*Random hyperplane hash family*) The random hyperplane hash family $\mathcal{H}_{\text{rh}}$ is the set of hash functions $\mathcal{H}_{\text{rh}} \triangleq \{h : \mathbb{R}^d \to \{0,1\}\}$, where $\mathbf{r}_h$ is a random d-dimensional Gaussian unit vector and $h(\mathbf{x}) \triangleq \begin{cases} 1 & \text{if } \mathbf{r}_h^T \mathbf{x} \geq 0 \\ 0 & \text{if } \mathbf{r}_h^T \mathbf{x} < 0 \end{cases}$.

This hash family is well-known to provide the property that the probability of two vectors colliding is a function of the degree of the angle between them [3]:

$$\Pr(h(\mathbf{x}_u) = h(\mathbf{x}_v)) = 1 - \frac{\theta(\mathbf{x}_u, \mathbf{x}_v)}{\pi} = 1 - \frac{\arccos(\mathbf{x}_u^T \mathbf{x}_v)}{\pi},$$

where the last equality holds due to normalized embeddings.

To lower the false-positive rate, it is conventional to form a new hash function by sampling b hash functions from $\mathcal{H}_{\text{rh}}$ and concatenating the hash codes: $g(\cdot) = (h_1(\cdot), h_2(\cdot), \ldots, h_b(\cdot))$. The new hash function is from another LSH family:

Definition 5 (*b-AND-random hyperplane hash family*) The b-AND-Random hyperplane hash family is the set of hash functions $\mathcal{H}_{\text{and}}^b \triangleq \{g : \mathbb{R}^d \to \{0,1\}^b\}$, where $g(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_b(\mathbf{x}))$ is formed by concatenating b randomly sampled hash functions $h(\cdot) \in \mathcal{H}_{\text{rh}}$ for some $b \in \mathbb{N}$.

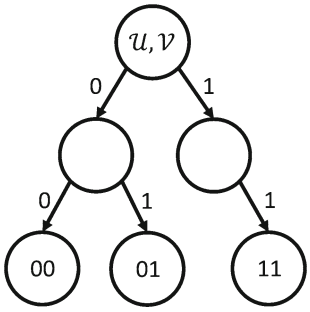


Fig. 2 LSH Tree. Each level of the tree corresponds to a random hash function, and the leaves correspond to buckets

Since the hash functions are sampled randomly from $\mathcal{H}_{\text{rh}}$,

$$\Pr(g(\mathbf{x}_u) = g(\mathbf{x}_v)) = \left(1 - \frac{\arccos(\mathbf{x}_u^T \mathbf{x}_v)}{\pi}\right)^b. \tag{13}$$

Only vectors that are not split by all b random hyperplanes end up with the same hash codes, so this process lowers the false-positive rate. However, it also increases the false-negative rate for the same reason. The conventional LSH scheme then repeats the process r times, computing the dot product exactly over all pairs that match in at least one b-dim hash code, in order to lower the false-negative rate. The challenge of this approach is determining how to set b. To do so, we first define the hash buckets of a hash function and their volume.

Definition 6 (*Hash buckets and volume*) Given an equivalence class $\mathcal{C}_i = \mathcal{V}_u \times \mathcal{V}_v$ and a hash function $g(\cdot) : \mathbb{R}^d \to \{0, 1\}^b$, after applying $g(\cdot)$ to all $v \in \mathcal{V}_u \cup \mathcal{V}_v$, a hash bucket

$$\beta = \{u \in \mathcal{V}_u, v \in \mathcal{V}_v : g(u) = g(v) = \beta_{\text{hashcode}}\}$$

consists of subsets $\mathcal{V}_u^{(\beta)} \subseteq \mathcal{V}_u, \mathcal{V}_v^{(\beta)} \subseteq \mathcal{V}_v$ of nodes that mapped to hashcode $\beta_{\text{hashcode}} \in \{0, 1\}^b$. The set of hash buckets $\mathcal{B}_g = \{\beta : |\beta| > 0\}$ consists of all nonempty buckets. We define the volume of the buckets as the number of pairs (u, v) where u and v landed in the same bucket:

$$\text{Vol}(\mathcal{B}_g) \triangleq |\{(u, v) : g(\mathbf{x}_u) = g(\mathbf{x}_v)\}| = \textstyle\sum_{\beta \in \mathcal{B}_g} |\mathcal{V}_u^{(\beta)} \times \mathcal{V}_v^{(\beta)}|.$$

Since we are after the κ closest pairs, we want to find a hash function $g(\cdot)$ such that $\text{Vol}(\mathcal{B}_g) \geq \kappa$. But since we want to search as few pairs as possible, we seek the value of b that minimizes $\text{Vol}(\mathcal{B}_g)$ for some $g(\cdot) \in \mathcal{H}_{\text{and}}^b$ subject to the constraint that $\text{Vol}(\mathcal{B}_g) \geq \kappa$.

Any hash function $g \in \mathcal{H}_{\text{and}}^b$ corresponds to a binary prefix tree, like Fig. 2. Each level of the tree corresponds to one $h \in \mathcal{H}_{\text{rh}}$, and the leaves correspond to the buckets $\mathcal{B}_g$. Thus, to automatically identify the best value of b, we can recursively grow the tree, branching each leaf with a new random hyperplane hash function $h \in \mathcal{H}_{\text{rh}}$, until $\text{Vol}(\mathcal{B}_g) < \kappa$, then undo the last branch. At that point, the depth of the tree equals b, and is the largest value such that $\text{Vol}(\mathcal{B}_g) \geq \kappa$. To prevent this process from repeating indefinitely in edge cases, we halt the branching at a maximum depth $b_{\max}$. This approach is closely related to LSH Forests [3], but with some key differences, which we discuss shortly.

Theorem 4 *Given a hash function $g \in \mathcal{H}_{and}^b$, the κ closest pairs in $\mathcal{C}_i$ are the κ most likely pairs to be in the same bucket:*

$$Pr(g(\mathbf{x}_u) = g(\mathbf{x}_v)) > Pr(g(\mathbf{x}_{u'}) = g(\mathbf{x}_{v'})) \iff \mathbf{x}_u^T \mathbf{x}_v > \mathbf{x}_{u'}^T \mathbf{x}_{v'}.$$

Proof Since $\arccos(x)$ is a *decreasing* function of x, Eq. (13) shows that $\Pr(g(\mathbf{x}_u) = g(\mathbf{x}_v))$ is an *increasing* function of $\mathbf{x}_u^T \mathbf{x}_v$. The result follows from this. □

While $\mathbf{x}_u^T \mathbf{x}_v > \mathbf{x}_{u'}^T \mathbf{x}_{v'}$ implies that (u, v) are *more likely* than (u', v') to be in the same bucket, it does not guarantee that this outcome will *always* happen. Thus, we repeat the process r times, creating r binary prefix trees and, searching the pairs that fall in the same bucket in any tree for the top κ. Setting the r parameter is considered of minor importance, as long as it is sufficiently large (e.g., 10) [3].

Each hash bucket (tree leaf) contains a sets of node pairs. Critically, each bucket contains this information only tacitly because it consists of two sets $\mathcal{V}_u^{(\beta)}$ and $\mathcal{V}_v^{(\beta)}$ of *nodes*, and the *node pairs* are only realized by taking the Cartesian product of these two sets. Moreover, the volume can be computed by multiplying the size of the respective sets (i.e., $|\mathcal{V}_u^{(\beta)} \times \mathcal{V}_v^{(\beta)}| = |\mathcal{V}_u^{(\beta)}| * |\mathcal{V}_v^{(\beta)}|$). This property is what allows the search to implicitly avoid distant pairs: each time a new branch is formed, the nodes that go left will no longer pair with those that go right, and only when the tree is fully grown are the pairs in the buckets (leaves) actually instantiated by computing the Cartesian products.

Differences from LSH Forests [3]. LSH Forests are designed for KNN-search, which seeks to return the nearest neighbors to a *query vector*. In contrast, our approach is designed for κ-closest pairs search, which seeks to return the κ closest pairs in a set $\mathcal{C}_i$. LSH Forests grow each tree until each vector is in its own leaf. We grow each tree until we reach the target bucket volume κ. LSH Forests allow variable length hash codes, since the nearest neighbors of different query vectors may be at different relative distances. All our leaves are at the same depth so that the probability of (u, v) surviving together to the leaf is an increasing function of their dot product.

4.3.2 Avoiding exhaustive search for heuristics

For the heuristic definitions of proximity in Eqs. (3)–(5), there are two approaches to solving Problem 3. The first is to construct embeddings from the CN and AA scores (this does not apply to JS). For CN, if we let the node embeddings be their corresponding rows in the adjacency matrix, i.e., $\mathbf{X}_{\text{CN}} = \mathbf{A}$, then $sim_{\text{CN}}(u, v) = \mathbf{x}_u^T \mathbf{x}_v$. Similarly, $\mathbf{X}_{\text{AA}} = \mathbf{A} \cdot 1/\sqrt{\log(\mathbf{D})}$, yields $sim_{\text{AA}}(u, v) = \mathbf{x}_u^T \mathbf{x}_v$, where $\mathbf{D}$ is a diagonal matrix recording the degree of each node. Thus, the LSH solution just described can be applied. The second approach uses the fact that all three heuristics are defined over the 1-hop neighborhoods of nodes (u, v). Thus, to have nonzero proximity, (u, v) must be within 2-hops of each other, and any pairs not within 2-hops can implicitly be ignored.

4.3.3 Bail out refinement

To this point we have assumed that the proximity model used in LinkWaldo is highly informative and accurate. However, in reality, heuristics may not be informative for all equivalence classes, and even learned, latent proximity models, can fail to encode adequate information. For instance, it is challenging to learn high-quality representations for low-degree nodes. Thus, we introduce a refinement to LinkWaldo that automatically identifies when a proximity model is uninformative in an equivalence class, and allows it to *bail out* of searching that equivalence class.

Proximity Model Error. The error that a proximity model makes is the probability $\Pr(sim(u, v) < sim(u', v'))$ that it gives a higher proximity for some unlinked pair $(u', v') \notin \mathcal{E}$ than for some linked pair $(u, v) \in \mathcal{E}$. This is what referred to as Typ3- Err in Sect. 3.4.

By this definition of error, we expect strong proximity models to mostly assign higher proximity between observed edges than future or missing edges: $\Pr(sim(u, v) > sim(u', v')) \approx 1$ for some $(u, v) \in \mathcal{E}$ and $(u', v') \in \mathcal{E}_{\text{new}}$. Thus, on our way to finding the top-κ most similar (unlinked) pairs in an equivalence class (Problem 3), we expect to encounter a majority of the *observed* edges (linked pairs) $|\mathcal{E} \cap \mathcal{C}_i|$ that fall in that class. For a user-specified error tolerance ζ, LinkWaldo will bail out and return no pairs from any equivalence class where less than ζ fraction of its observed edges are encountered on the way to finding the κ most similar unlinked pairs. LinkWaldo keeps track of how many pairs were skipped by bailing out, and replaces them (after step **S4**) by adding to $\mathcal{P}$ the top-ranked pairs of a heuristic (e.g., AA).

4.4 (S4) Augmenting pairs from global pool

Since LinkWaldo returns a standard deviation below the expected number of new pairs in each equivalence class, it chooses the remaining pairs up to k from $\tilde{\mathcal{P}}_G$. To do so, it considers pairs in descending order on the input similarity function $sim(\cdot, \cdot)$ and greedily adds to $\mathcal{P}$ until $|\mathcal{P}| = k$.

4.5 Complexity analysis

Let γ be the time complexity of the node grouping (**S1**). Computing the expected number of new edges in each cell and the variances directly from the observed links, (**S2**), is $O(m)$. The complexity of searching equivalence classes (**S3**) comes from hashing each node in the decomposition $O(b_{\max})$ times and finding the κ_i closest pairs in the $O(\kappa_i)$ pairs that land in the same bucket: $\sum_{\mathcal{C}_i \in \Pi} O(|\mathcal{V}_u \cup \mathcal{V}_v| b_{\max} + \kappa_i) = O(n b_{\max} + k)$. This assumes that we do not encounter unrealistic scenarios, such as the embeddings being equivalent and hence inseparable, and that $b_{\max}$ is set large enough that the volume of tree leaves is not asymptotically larger than $O(\kappa_i)$. Adding from the global pool (**S4**) takes $O(k)$ time, since $|\tilde{\mathcal{P}}_G| = O(k)$ and can be maintained in sorted order in similar fashion to the merge in merge sort. Thus, the total time complexity is $O(\gamma + m + n b_{\max} + k)$.

5 Evaluation

We evaluate LinkWaldo on four research questions:

- (**RQ1**) Does the set $\mathcal{P}$ returned by LinkWaldo have high recall and precision?
- (**RQ2**) Is LinkWaldo scalable?
- (**RQ3**) How do parameters affect performance?
- (**RQ4**) What are the main sources of error?

Before we answer these questions, we describe the data and empirical setup.
Data We evaluate LinkWaldo on a large, diverse set of networks: metabolic, social, communication, and information networks (Table 2). Moreover, we include datasets to evaluate in both LP scenarios: (1) returning possible *missing* links in static graphs and (2) returning possible *future* links in temporal graphs. We treat all graphs as undirected.

- *Metabolic networks.* Yeast [40], HS-Protein [16], and Protein-Soy [18] are metabolic protein networks, where edges denote known associations between proteins in different species. Yeast contains proteins in a species of yeast, HS-Protein in human beings, and Protein-Soy in Glycine max (soybeans).

Table 2 Dataset properties and statistics: if the graph is temporal or static, density, degree assortativity [24], and number of nodes and edges

Graph	Temporal	Density (%)	Assortativity	Nodes n	Edges m
Yeast	–	0.41%	0.4539	2375	11, 693
DBLP	–	0.06	−0.0458	12, 595	49, 638
Facebook1	–	1.08	0.0636	4041	88, 235
MovieLens	✓	2.90	−0.2268	2627	100, 000
HS-Protein	–	0.74	0.2483	6329	147, 548
arXiv	–	0.11	0.2051	18, 772	198, 110
MathOverflow	✓	0.06	−0.1979	24, 820	199, 974
Enron	✓	0.01	−0.1667	87, 275	299, 221
Reddit	✓	0.01	−0.1278	67, 180	309, 667
Epinions	–	0.01	−0.0406	75, 881	405, 741
Facebook2	✓	0.04	0.1770	63, 733	817, 063
Digg	✓	< 0.01	−0.0557	279, 376	1, 546, 541
Protein-Soy	–	1.64	−0.0192	45, 116	16, 691, 679

- *Social networks.* Facebook1 [18] and Facebook2 [16] capture friendships on Facebook, Reddit [18] encodes links between subreddits (topical discussion boards), edges in Epinions [16] connect users who trust each other's opinions, MathOverflow [18] captures comments and answers on math-related questions and comments (e.g., user u answered user v's question), Digg [30] captures friendships among users.
- *Communication networks.* Enron [16] is an email network, capturing emails sent during the collapse of the Enron energy company.
- *Information networks.* DBLP [16] is a citation network, and arXiv [18] is a co-authorship network of Astrophysicists. MovieLens [16] is bipartite graph of users rating movies for the research project MovieLens. Edges encode users and the movies that they rated.

Setup Training Graph and Ground Truth. While using LinkWaldo in practice does not require a test set, in order to know how effective it is, we must *evaluate* it on ground truth missing links. As ground truth, we remove 20% of the edges. In the static graphs, we remove 20% at random. In the temporal graphs, we remove the 20% of edges with the most recent timestamps. If either of the nodes in the removed edge is not present in the training graph, we discard the edge from the ground truth. The graph with these edges removed is the training graph, which LinkWaldo and the baselines observe when choosing the set of unlinked pairs to return.

Method Configuration. We discuss in Sect. 5.3 how we choose which groupings to use and how many groups in each. Whenever used, we implement SG and CG by clustering embeddings with KMeans: xNetMF [12] and NetMF [28] (window size 1), respectively. In LSH, we set the maximum tree depth dynamically based on the size of an equivalence class: $b_{\max} = 12$ if $|\mathcal{C}_i| < 1B$, $b_{\max} = 15$ if $|\mathcal{C}_i| < 10B$, $b_{\max} = 20$ $|\mathcal{C}_i| < 25B$, $b_{\max} = 30$ otherwise. We set the number of trees r based on the fraction of $|\mathcal{C}_i|$ that we seek to return: $r = 5$ if $\kappa/|\mathcal{C}_i| < 0.0001$, $r = 10$ if $\kappa/|\mathcal{C}_i| < 0.001$ and $r = 25$ otherwise.

5.1 (RQ1) Recall and precision

Task setup We evaluate how effectively LINKWALDO returns in $\mathcal{P}$ the ground truth missing links, at values of k much smaller than n^2. We report k, chosen based on dataset size, in Table 3, and discuss effects of the choice in Sect. 5.3.1. We compare the set LINKWALDO returns to those of five baselines and evaluate both LINKWALDO-D, which uses grouping DG, and LINKWALDO-M, which uses DG, SG, and CG together. In both LINKWALDO variants, we consider the following proximities (Sect. 3.2) as input and report the results that are best: LaPM using NETMF [28] embeddings (window sizes 1 and 2), and AA, the best heuristic proximity. For the bipartite MovieLens, we use BINE [9], an embedding method designed for bipartite graphs. We report the input proximity model for each dataset in Table 6 in appendix. We set the exact-search and bailout tolerances to $\tau = 25\,M$ and $\zeta = 0.5$, which we determined via a parameter study in Sect. 5.3. Results are averages over five random seeds (Sect. 5): for static graphs, the randomly removed edges are different for each seed; for temporal graphs, the latest edges are always removed, so the LSH hash functions are the main source of randomness.

Metrics We use Recall (R@k), the fraction of known missing/future links that are in the size-k set returned by the method, and Precision (P@k), the fraction of the k pairs that are known to be missing/future links. Recall is a more important metric, since (1) the returned set of pairs $\mathcal{P}$ does not contain final predictions, but rather pairs for a LP method to make final decisions about, and (2) our real-world graphs are inherently incomplete, and thus pairs returned that are not *known* to be missing links, could nonetheless be missing in the original dataset prior to ground truth removal (i.e., the open-world assumption [32]). We report both in Table 3.

Baselines We use five baselines. **NMF+BAG** [8] uses nonnegative matrix factorization (NMF) and a bagging ensemble to return k pairs while pruning the search space. We use their reported strongest version: the *Biased Edge Bagging* version with *Node Uptake* and *Edge Filter* optimizations (*Biased(NMF+)*). We use the authors' recommended parameters when possible: $\epsilon = 1$, $\mu = 0.1$, $f = 0.1$, $\rho = 0.75$, number of latent factors $d = 50$, and ensemble size μ/f^2. In some cases, these suggested parameters led to fewer than k pairs being returned, in which case we tweaked the values of ϵ, μ, and f until k were returned. We report these deviations in Table 6 in appendix. We use our own implementation.

We also use four proximity models, which we showed to be special cases of FLLM in Sect. 3.5: **LaPM** ranks pairs *globally* based on the dot product of their embeddings and returns the top k. To avoid searching all-pairs, we use the same LSH scheme that we introduce in Sect. 4.3 for LINKWALDO. We set $r = 25$, and like LINKWALDO, use NETMF with a window size of 1 or 2, except for MovieLens, where we use BINE.

JS, **CN**, and **AA** are defined in 3.2. We exploit the property described in 4.3.2—i.e., all these scores are zero for nodes beyond two hops. We compute the scores for all nodes within two hops and return the top k unlinked pairs.

Results Across the 13 datasets, LINKWALDO is the best performing method on 10, in both recall and precision. The LINKWALDO-M variant is slightly stronger than LINKWALDO-D, but the small gap between the two demonstrates that even simple node groupings can lead to strong improvements over baselines. LINKWALDO generalizes well across the diverse types of networks. In contrast, the heuristics perform well on social networks, but not as well on, e.g., metabolic networks (Yeast, HS-Protein, and Protein-Soy). Furthermore, the heuristic baselines cannot extend to bipartite graphs like MovieLens, because fundamentally, all links form between nodes more than one hop away. These observations demonstrate the value of learning from the observed links, which LINKWALDO does via resemblance. We also observe

Table 3 On each dataset, we highlight the cell of the top-performing method with bold text and italics, and the second best with bold text only

Dataset	*Metric*	NMF+BAG [8]	LaPM	JS	CN	AA	LINKWALDO-D	LINKWALDO-M
Yeast	R@10K	0.4078 ± 0.01	0.4400 ± 0.01	0.4766 ± 0.01	0.6142 ± 0.01	0.6590 ± 0.01	**0.6762 ± 0.01**	***0.6926**** ± ***0.01***
0.39%	P@10K	0.0898 ± 0.00	0.0969 ± 0.00	0.1049 ± 0.00	0.1352 ± 0.00	0.1451 ± 0.00	**0.1489 ± 0.00**	***0.1525**** ± ***0.00***
DBLP	R@100K	0.2319 ± 0.00	0.0927 ± 0.00	0.0389 ± 0.00	0.3379 ± 0.00	0.3775 ± 0.00	**0.4270 ± 0.00**	***0.4271**** ± ***0.00***
0.15%	P@100K	0.0204 ± 0.00	0.0082 ± 0.00	0.0034 ± 0.00	0.0298 ± 0.00	0.0332 ± 0.00	***0.0376**** ± ***0.00***	***0.0376**** ± ***0.00***
Facebook1	R@100K	0.4036 ± 0.02	0.8005 ± 0.00	0.8244 ± 0.00	0.8547 ± 0.00	0.8863 ± 0.00	**0.8975 ± 0.00**	***0.9059**** ± ***0.00***
1.34%	P@100K	0.0711 ± 0.00	0.1410 ± 0.00	0.1453 ± 0.00	0.1506 ± 0.00	0.1562 ± 0.00	**0.1581 ± 0.00**	***0.1596**** ± ***0.00***
MovieLens	R@100K	0.1221 ± 0.02	0.2096 ± 0.01	0.0000 ± 0.00	0.0000 ± 0.00	0.0000 ± 0.00	**0.1667 ± 0.01**	***0.3662**** ± ***0.01***
3.57%	P@100K	0.0035 ± 0.00	0.0060 ± 0.00	0.0000 ± 0.00	0.0000 ± 0.00	0.0000 ± 0.00	**0.0048 ± 0.00**	***0.0105**** ± ***0.00***
HS-Protein	R@100K	0.5127 ± 0.02	0.4033 ± 0.01	0.4768 ± 0.00	0.7998 ± 0.00	0.8429 ± 0.00	**0.8878 ± 0.00**	***0.9038**** ± ***0.00***
0.52%	P@100K	0.1506 ± 0.00	0.1185 ± 0.00	0.1400 ± 0.00	0.2349 ± 0.00	0.2476 ± 0.00	**0.2608 ± 0.00**	***0.2655**** ± ***0.00***
arXiv	R@100K	0.2877 ± 0.00	0.2584 ± 0.00	0.5149 ± 0.00	0.5576 ± 0.00	0.6539 ± 0.00	**0.7004 ± 0.00**	***0.7032**** ± ***0.00***
0.06%	P@100K	0.2017 ± 0.00	0.1812 ± 0.00	0.3610 ± 0.00	0.3909 ± 0.00	0.4585 ± 0.00	**0.4911 ± 0.00**	***0.4930**** ± ***0.00***
MathOverflow	R@1 M	0.3901 ± 0.00	0.0279 ± 0.00	0.0084 ± 0.00	0.4201 ± 0.00	***0.4333**** ± ***0.00***	**0.4227 ± 0.00**	0.4233 ± 0.00
0.55%	P@1 M	0.0070 ± 0.00	0.0005 ± 0.00	0.0001 ± 0.00	0.0075 ± 0.00	***0.0078**** ± ***0.00***	**0.0076 ± 0.00**	**0.0076 ± 0.00**

Table 3 continued

Dataset	*Metric*	NMF+BAG [8]	LaPM	JS	CN	AA	LINKWALDO-D	LINKWALDO-M
Enron	R@1 M	0.2551 ± 0.00	0.0008 ± 0.00	0.0004 ± 0.00	0.3071 ± 0.00	***0.3209**** ± ***0.00***	**0.3166 ± 0.00**	0.3080 ± 0.00
0.04%	P@1 M	0.0089 ± 0.00	0.0000 ± 0.00	0.0000 ± 0.00	0.0107 ± 0.00	***0.0112**** ± ***0.00***	**0.0111 ± 0.00**	0.0108 ± 0.00
Reddit	R@1 M	0.3165 ± 0.00	0.0015 ± 0.00	0.0006 ± 0.00	0.3796 ± 0.00	0.4038 ± 0.00	**0.4096 ± 0.00**	***0.4186**** ± ***0.00***
0.06%	P@1 M	0.0130 ± 0.00	0.0001 ± 0.00	0.0000 ± 0.00	0.0156 ± 0.00	0.0166 ± 0.00	**0.0168 ± 0.00**	***0.0172**** ± ***0.00***
Epinions	R@1 M	0.3312 ± 0.00	0.0371 ± 0.00	0.0383 ± 0.00	0.3871 ± 0.00	0.4291 ± 0.00	**0.4292 ± 0.00**	***0.4323**** ± ***0.00***
0.04%	P@1 M	0.0242 ± 0.00	0.0027 ± 0.00	0.0028 ± 0.00	0.0283 ± 0.00	**0.0313 ± 0.00**	**0.0313 ± 0.00**	***0.0316**** ± ***0.00***
Facebook2	R@1 M	0.0948 ± 0.00	0.1947 ± 0.00	0.3605 ± 0.00	0.2439 ± 0.00	0.2832 ± 0.00	***0.3796**** ± ***0.00***	**0.3776 ± 0.00**
0.06%	P@1 M	0.0145 ± 0.00	0.0298 ± 0.00	0.0551 ± 0.00	0.0373 ± 0.00	0.0433 ± 0.00	***0.0580**** ± ***0.00***	**0.0577 ± 0.00**
Digg	R@10 M	0.2459 ± 0.00	0.0035 ± 0.00	0.0015 ± 0.00	0.2952 ± 0.00	***0.3066**** ± ***0.00***	**0.3053 ± 0.00**	0.3052 ± 0.00
0.04%	P@10 M	0.0032 ± 0.00	0.0000 ± 0.00	0.0000 ± 0.00	0.0038 ± 0.00	***0.0040**** ± ***0.00***	***0.0040**** ± ***0.00***	***0.0040**** ± ***0.00***
Protein-Soy	R@10 M	0.3624 ± 0.02	0.1225 ± 0.00	0.2792 ± 0.00	0.3573 ± 0.00	0.3636 ± 0.00	**0.5781 ± 0.00**	***0.6016**** ± ***0.03***
0.99%	P@10 M	0.2178 ± 0.01	0.0736 ± 0.00	0.1678 ± 0.00	0.2147 ± 0.00	0.2185 ± 0.00	**0.3473 ± 0.00**	***0.3615**** ± ***0.02***
	Avg R	0.3048	0.1994	0.2323	0.4273	0.4585	**0.5074**	***0.5281***
	Avg P	0.0635	0.0506	0.0754	0.0969	0.1056	**0.1213**	***0.1238***

Bold denotes the best and second-best performing models

An "*" denotes statistical significance at a 0.05 p-value in a paired t-test. The "**" means that the difference between the better performing variant of LINKWALDO was also significantly better than the other variant at the same p-value. Under each dataset, we give the percentage of the quadratic search space that the value of k corresponds to (usually $< 1\%$). On average, LINKWALDO-M is the best-performing method and LINKWALDO-D the second best

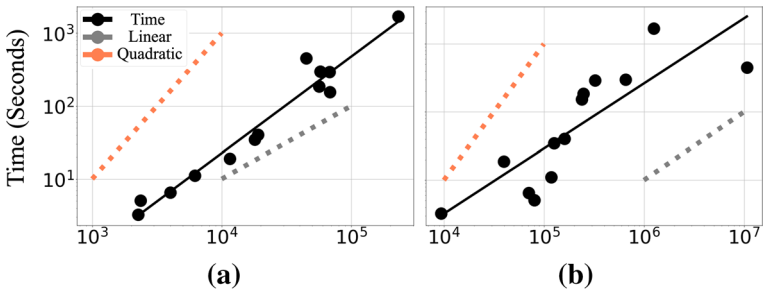


Fig. 3 LINKWALDO is sub-quadratic on the number of nodes (**a**) and linear on the number of edges (**b**)

that heuristic definitions of similarity, such as AA, outperform latent embeddings (LaPM) that capture proximity. We conjecture that the embedding methods are more sensitive to the massive skew of the data, because even random vectors in high-dimensional space can end up with some level of proximity, due to the curse of dimensionality. This suggests that the standard approach of evaluating on a balanced test set may artificially inflate results.

In the three datasets where LINKWALDO does not outperform AA, it is only outperformed by a small margin. Furthermore, the four datasets with the largest total Variation distances between p_n and p_o are MovieLens, MathOverflow, Enron, and Digg. Theorem 2 suggests that LINKWALDO may incur the most error in these datasets. Indeed, these are the only three datasets where LINKWALDO fails to outperform all other methods (with MovieLens being bipartite, as discussed above). While the performance on temporal networks is strong, the higher total Variation distance suggests that the assumption that $p_o = p_n$ may sometimes be violated due to *concept drift* [5]. Thus, a promising future research direction is to use the timestamps of observed edges to predict roadmap drift over time, in order to more accurately estimate the future roadmap.

5.2 (RQ2) Scalability

Task setup We evaluate how LINKWALDO scales with the number of edges, and the number of nodes in a graph by running LINKWALDO with fixed parameters on all datasets. We set $k = 1\,M$, use NETMF (window-size of 1) as $sim(\cdot, \cdot)$, and do not perform bailout ($\zeta = 0$). All other parameters are identical to **RQ1**. We use our Python implementation on an Intel(R) Xeon(R) CPU E5–2697 v3, 2.60GHz with 1TB RAM.
Results The results in Fig. 3 demonstrate that in practice, LINKWALDO scales linearly on the number of edges, and sub-quadratically on the number of nodes.

5.3 (RQ3) Parameter analysis

Setup We evaluate the quality of different groupings (Sect. 4.1), and how the number of groups in each affects performance. On four graphs, Yeast, arXiv, Reddit, and Epinions, we run LINKWALDO with groupings DG, SG, and CG, varying the number of groups $|\Gamma| \in \{5, 10, 25, 50, 75, 100\}$. We also investigate pairs of groupings, and the combination of all three groupings, via grid search of the number of groupings in each. We also evaluated $\tau \in \{1M, 10M, 25M, 50M\}$, the tolerance for searching equivalence classes exactly vs. approximately with LSH, and $\zeta \in \{0, 0.1, 0.25, 1/3, 0.5, 2/3\}$, the fraction of training pairs we allow the proximity function to miss before we bailout of an equivalence class.

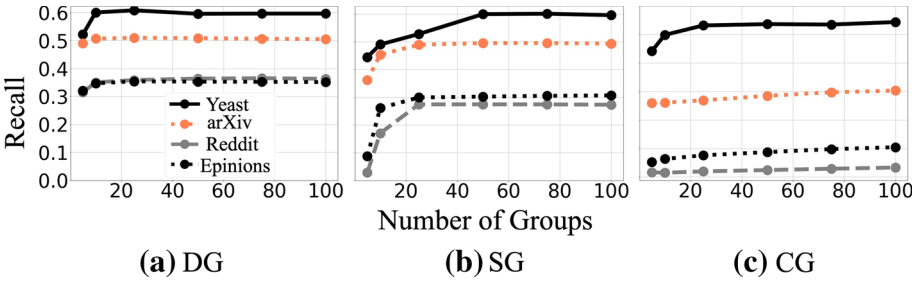


Fig. 4 Number of groups for groupings DG (**a**), SG (**b**), and CG (**c**)

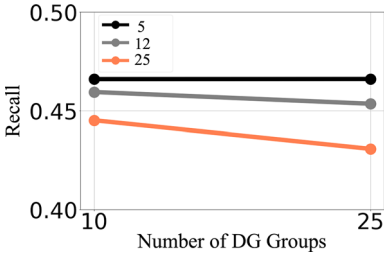


Fig. 5 Parameter analysis: recall for different combinations of number of log-bins for DG (10 or 25, shown on the x-axis), and number of groups for SG and CG (the same number is used for both; each line corresponds to a different number). Using 5 groups for SG and CG (black line) is best and slightly better when 25 groups are used for DG

Results Varying the groupings. The results for the individual groupings are shown in Fig. 4. Grouping by log-binning nodes based on their degree, (i.e., DG) is in general the strongest grouping. Across all three groupings, we find that $|\Gamma| = 25$ is a good number of groups. We found that using all three groupings was the best combination, with 25 log-bins, 5 structural clusters, and 5 communities. This is shown in Fig. 5, which varies the number of DG groups on the x-axis and each line represents different choices for the number of SG and CG groups. Five groups for SG and CG (black line) had slightly higher recall with 25 DG groups. For individual groupings, we observe diminishing returns, and in multiple groupings, slightly diminished performance when the number of groups in each grows large. We omit the figures for τ and ζ, but found that $\tau = 25\,M$ and $\zeta = 0.5$ were the best parameters.

Varying τ. When we vary $\tau \in \{1M, 10M, 25M, 50M\}$, we expect LinkWaldo's time to follow a U-shaped curve and its recall to show diminishing returns. The smaller τ, the fewer equivalence classes we will search exactly. LSH has overhead due to needing to repeatedly hash the node pairs in an equivalence class, and due to redundancy across trees. This overhead is best avoided by not setting τ too small. On the other hand, if τ is too large, the quadratic complexity of searching large equivalence classes could dwarf the LSH overhead. This is why we expect to see a U-shaped curve with respect to runtime. Because LSH is approximate, it could introduce a small amount of error, in which case we expect recall to slowly increase as τ gets larger (i.e., as more equivalence classes are searched exactly). Thus, our goal is to find the largest value of τ that is safely in the bottom of the U-shaped time curve. In Fig. 6b, the recall is stable across all values of τ, indicating that our locality-sensitive hashing approach does not degrade macro-level performance. The U-shaped curve in Fig. 6a becomes apparent only for large graphs (Digg in this case), because the cost of exhaustively searching an equivalence class only becomes problematic for large graphs. The curve starts to increase rapidly at $\tau = 50\,M$, so $\tau = 25M$ is the largest value in the bottom of the U-shaped curve.

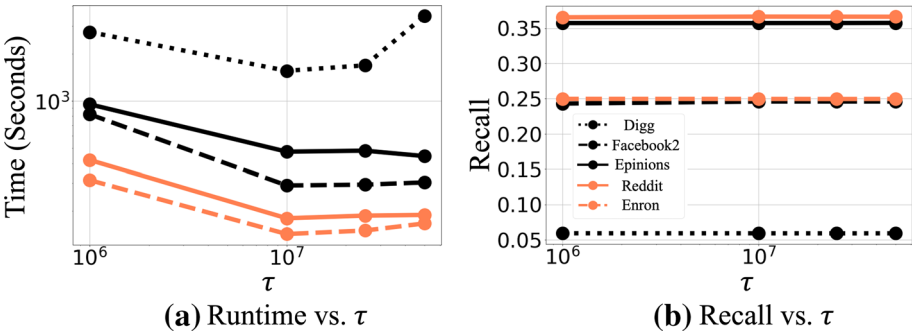


Fig. 6 Runtime and recall for varying values of tolerance τ. **a** The U-shaped curve, which is apparent for the large graph Digg, starts to increase rapidly at $\tau = 50M$, so $\tau = 25M$ is the largest value in the bottom of the curve. **b** The recall is stable across all values of τ, indicating that our locality-sensitive hashing approach does not degrade macro-level performance

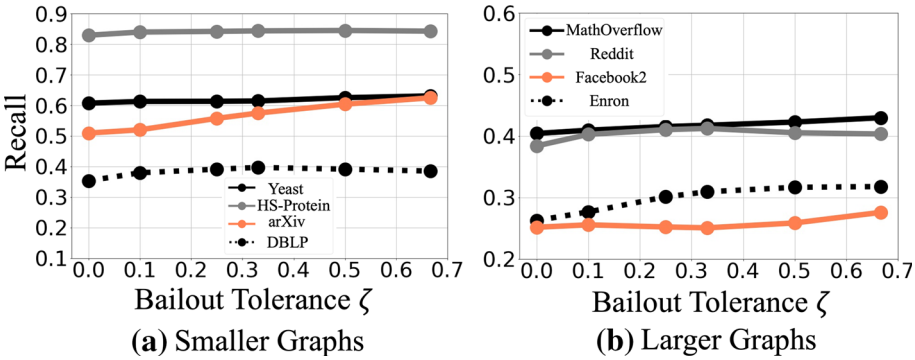


Fig. 7 Recall for varying bailout tolerance ζ for small (**a**) and large (**b**) graphs. The recall improves with larger ζ up to a point (diminishing returns), so we chose $\zeta = 0.5$ for our experiments

We choose this value for the rest of our experiments, but $\tau = 10M$ would also be a sensible choice.

Varying ζ. In our parameter analysis, we vary $\zeta \in \{0, 0.1, 0.25, 1/3, 0.5, 2/3\}$. In Fig. 7, we plot the results for four smaller graphs on the left (a) and four larger graphs on the right (b) in order to visualize the trends more clearly. The figure shows that the recall improves with larger ζ up to a point, but in some cases begins to degrade for $\zeta > 0.5$. Thus, we choose $\zeta = 0.5$ in our experiments.

5.3.1 Effect of *k*

Setup We now evaluate the effects of budget k on recall. We ran LINKWALDO-M with the same parameters as in the main performance experiment in Sect. 5.1, but over multiple values of k. We compared the performance to AA and NMF+BAG, the best performing baselines in Sect. 5.1. For this experiment, we used HS-Protein and Facebook2. The values of k that we chose were roughly up to an order of magnitude greater than m.

Results Table 4 gives results (averages over 3 seeds) for multiple values of k. For reasonable values of k results are mostly stable. The main exception is for small k, where NMF+BAG performs well in some cases. This is consistent with its design: to return a small, accurate set of top-k predictions, rather than a candidate set.

Table 4 We report "$< k$" if fewer than k pairs are returned

Metric	HS-Protein			Facebook2		
	NMF+BAG	AA	LINKWALDO-M	NMF+BAG	AA	LINKWALDO-M
R@10K	**0.2629**	0.1909	0.2251	0.0093	0.0103	**0.0172**
R@100K	$< k$	0.8443	**0.9035**	0.0422	0.0672	**0.1106**
R@1M	$< k$	0.9747	**0.9847**	0.0970	0.2832	**0.3781**
R@5M	N/A	N/A	N/A	$< k$	0.5561	**0.5762**

Bold denotes the best-performing model
N/A indicate values of k outside the scale of the dataset

Table 5 Average Pearson correlation (over three runs) between TYP2- ERR and recall (R), and TYP3- ERR and recall for different datasets

Graph	Pearson r TYP2- ERR with R	Pearson r TYP3- ERR with R
Yeast	$r = -0.6378$ ($p = 0.36$)	$r = \mathbf{-0.9945}$ ($p = 0.01$)
DBLP	$r = 0.0665$ ($p = 0.93$)	$r = \mathbf{-0.9998}$ ($p < 0.01$)
Facebook1	$r = -0.4284$ ($p = 0.57$)	$r = \mathbf{-0.9774}$ ($p = 0.02$)
HS-Protein	$r = -0.8493$ ($p = 0.15$)	$r = \mathbf{-0.9997}$ ($p < 0.01$)
arXiv	$r = -0.3586$ ($p = 0.64$)	$r = \mathbf{-0.9991}$ ($p < 0.01$)
MathOverflow	$r = 0.2478$ ($p = 0.75$)	$r = \mathbf{-0.9997}$ ($p < 0.01$)
Enron	$r = 0.1763$ ($p = 0.82$)	$r = \mathbf{-0.9992}$ ($p < 0.01$)
Reddit	$r = 0.1050$ ($p = 0.9$)	$r = \mathbf{-0.9997}$ ($p < 0.01$)
Epinions	$r = 0.1763$ ($p = 0.82$)	$r = \mathbf{-0.9994}$ ($p < 0.01$)

Bold denotes the best predictor of recall
TYP3- ERR is a significantly better predictor of recall than TYP2- ERR

5.4 (RQ4) Sources of error

In Sect. 3.4, we discussed three types of error. We now analyze how much these contribute to reduced recall in practice. As discussed before, TYP1- ERR can be avoided by setting k higher, so we focus only on TYP2- ERR and TYP3- ERR.

Setup We ran LINKWALDO for each of the four groupings (DG, SG, CG, and MG) and compute: (1) the TYP2- ERR and TYP3- ERR error, and (2) the resulting recall. We then computed the Pearson correlation between the TYP2- ERR error and Recall, and the TYP3- ERR error and Recall. Negative correlation is expected, since it means that increased error corresponds to decreased recall. We ran this experiment with bailout turned off, since this would interfere with our interpretation of TYP3- ERR. We set k high enough to ensure no TYP1- ERR.

Results We show results from nine datasets in Table 5, which are each averages over three runs. TYP3- ERR is strongly negatively correlated with recall, while TYP2- ERR is not. The TYP3- ERR correlations are all statistically significant at a 0.05 significance level, and 7/9 are at 0.01. On the other hand TYP2- ERR is never significantly correlated. Thus, TYP3- ERR is a better predictor of recall than TYP2- ERR error. Results were very similar when using Spearman's ρ correlation, which does not assume a linear relationship.

As discussed in Sect. 3.4, TYP2- ERR occurs when FLLM's assumption that future links follow the same distribution as observed links are violated. TYP3- ERR error occurs when the proximity model used within equivalence classes returns the wrong pairs. Taken together,

this suggests that FLLM is quite robust, and it may be fruitful to investigate improvements to proximity models.

While investigating the errors that LinkWaldo makes, we found one pervasive source of Typ3- Err error: equivalence classes with many low-degree nodes. In these equivalence classes, the proximity models struggled to return the correct pairs (i.e., those that were actually missing). This was exacerbated when the equivalence class was assigned a large fraction of the budget. This is straightforward to interpret in the context of the DG grouping: when many observed links are between a low degree node and a high degree node, high Typ3- Err error will ensue. Critically, the error is *not* occurring because of poor assignment of the budget—FLLM was correct to assign these equivalence classes large fractions of the budget because there were in fact many missing links in these equivalence classes. Rather, the error occurred because the proximity model failed to return many of those missing links, and instead returned others. This scenario is pervasive because of two facts. First, graphs tend to exhibit skewed degree distributions, which means that most links must involve low-degree nodes. Second, low-degree nodes are the most challenging for heuristic and embedding-based models to handle because their contexts are so sparse.

6 Discussion: real-world applications & future research

We foresee our work providing new opportunities for research in biology, particularly in mapping metabolic networks and connectomes. We focus here on connectomes, but most of what we say generalizes to metabolic networks.

Background. Connectomics [34] is the construction and study of complete mappings of organisms' nervous systems (*connectomes*). The only organism whose connectome has been completely mapped is the *C. elegans* nematode [38]. Despite the *C. elegans* nervous system containing only 302 neurons, it took decades for scientists to map the roughly 7,000 synaptic connections. The number of neurons grows rapidly with organism complexity: the fruit fly *Drosophila* contains 135,000 neurons, a mouse's hippocampus alone contains roughly a million neurons [2], and the human connectome is estimated to contain a hundred billion (10^{11}) neurons [34]. Due to its massive size, completing the human connectome at the level of individual neurons is generally considered implausible. Research instead considers the *macroscale* of brain regions and pathways, and the *mesoscale*, often consisting of groups of roughly a hundred neurons [34]. Either of these more granular levels of description lowers the number of nodes in the connectome by orders of magnitude, which makes the problem at least plausible.

However, as scientists seek to map a connectome, they face enormous numbers of possible synaptic connections. Each possible connection involves experimentation to determine whether or not the connection really exists. For instance, the nematode *C. elegans* would require $\binom{302}{2} = 45,451$ experiments to map the connectome by brute force; the fruit fly *Drosophila* would require $\binom{135,000}{2} \approx 9$ billion experiments.

A shortcoming of standard link prediction. A link predictor could be used to prioritize the most plausible connections. While a link predictor might make quick work of 9 billion predictions, scientists would still need to confirm the predictions via experimentation. Suppose that an underlying graph has—quite realistically—density $\delta = 0.001$, meaning that only 0.1% of neuron pairs form a synaptic connection, while 99.9% of the pairs will in fact *not* link. Even a highly accurate link predictor, for instance one with an AUC of 0.95, will return an enormous number of false-positive experiments to the scientist. To see this, observe that

AUC can be interpreted as the probability that a random synaptic connection is ranked higher than a random nonconnected pair of neurons by the link predictor. Thus, for an arbitrary link (positive), the expected number of nonlinks (negatives) ranked higher than it will be $(0.999 * 9,000,000,000) * 0.05 \approx 450\,M$ for a 0.95 AUC link predictor. Even though the fraction of negatives ranked above the positive is small (5%), the absolute number is enormous due to the sparsity of the graph. This reality is not captured in the standard link prediction setup with a balanced test set. Moreover, this problem arises not just in the evaluation of a link predictor but also in its use: even if evaluation takes sparsity into account, a scientist will still have an infeasible 450 M experiments to carry out.

The solution of candidate node-pair selection. We thus believe our problem formulation (Problem 3, reworded below) is more suited for network completion tasks faced in practice.

Problem 4 *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a budget $k << n^2$, return a set of plausible candidate node pairs $\mathcal{P} \subset \mathcal{V} \times \mathcal{V}$ of size $|\mathcal{P}| = k$ for further experimentation.*

The budget k can be set by the scientist, based on what is a feasible number of experiments to carry out.

LinkWaldo takes steps toward alleviating the challenges discussed above to solve Problem 4. Specifically, it combines the notion of *structural resemblance* and the common notion of *node proximity*. As a result, the node pairs deemed plausible will be those that are in close proximity *and* structurally resemble observed links—a subset of those that are just in close proximity. The benefits of this insight are demonstrated empirically in LinkWaldo's improved recall over baselines (Table 3).

Future Directions. Our proposed approach provides a way to tackle some of the challenges that come up in real scenarios, and also opens up several future directions.

First of all, on many datasets LinkWaldo's recall, despite significant improvement over baselines, is still low in absolute terms. This raises the question, *are there properties beyond structural resemblance and node proximity that can further handle the skew of the search space?*

Second, future research could probe the theoretical limits of our problem formulation: *given a target recall R and probability p, for what budgets k is it possible to return a size-k set of pairs that achieves recall R with probability at least p*? An answer to this question would allow, for instance, Connectomics researchers to know how many experiments they need to carry out (k) to complete the connectome ($R = 1$) and be confident (with probability p) that the graph is indeed complete. Moreover, the scientists could use this knowledge to determine a feasible scale for a particular connectome (neuron level, mesoscale, or macroscale). This can be further generalized to other scientific problems (e.g., metabolic network inference/completion).

7 Conclusion

In this paper, we focus on the under-studied and challenging problem of identifying a moderately sized set of node pairs for a link prediction method to make decisions about. We mitigate the vastness of the search space, filled with mostly nonlinks, by considering not just proximity, but also how much a pair of nodes *resembles* observed links. We formalize this idea in the Future Link Location Model, show its theoretical connections to stochastic block models and proximity models, and introduce an algorithm, LinkWaldo, that leverages it to return high-recall candidate sets, with only a tiny fraction of all pairs. Via our resemblance insight, LinkWaldo's strong performance generalizes from social networks to protein net-

works. Future directions include investigating the directionality of links, since the roadmap can incorporate this information, and extending to heterogeneous graphs with many edge and node types, like knowledge graphs.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10115-021-01632-x.

Acknowledgements This work is an invited extension of a paper accepted at ICDM 2020 [4] and is supported by an NSF Graduate Research Fellowship, NSF CAREER Grant No. IIS 1845491, Army Young Investigator Award No. W9–11NF1810397, and an Amazon faculty award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding parties.

A Proximity models

Here we discuss the proximity models used in Sect. 5.1. The proximity model for each method (along with parameters for the NMF+BAG baseline) is given in Table 6. We observe that the AA heuristic performed well on several datasets. During development, we tried over two dozen embedding methods (including GNNs) and found NETMF to be the most consistently strong for both LaPM and LINKWALDO. This, combined with the strong performance of AA, suggests that there is much room for improving proximity-preserving embedding methods. Improving their performance for low-degree nodes is of particular importance, as discussed in Sect. 5.4.

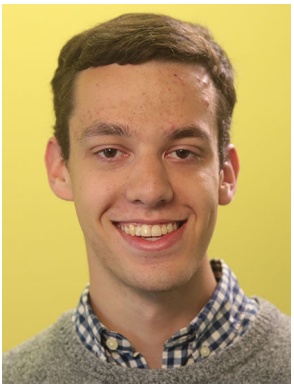
Table 6 The best input proximity model for LaPM and LINKWALDO (used in Sect. 5.1) and parameter deviations from default for NMF+BAG

Graph	LaPM	LINKWALDO-D	LINKWALDO-M	NMF+BAG
Yeast	NETMF- 2	NETMF- 2	NETMF- 2	$\epsilon = 0.5$
DBLP	NETMF- 2	NETMF- 2	NETMF- 2	Default
Facebook1	NETMF- 1	AA	AA	$\epsilon = 0.1$
MovieLens	BINE	BINE	BINE	$\epsilon = 0.05$
HS-Protein	NETMF- 2	NETMF- 2	NETMF- 2	$\epsilon = 0.75$
arXiv	NETMF- 2	AA	AA	Default
MathOverflow	NETMF- 2	NETMF- 1	NETMF- 1	$\epsilon, \mu, f = 0.5, 0.3, 0.3$
Enron	NETMF- 2	NETMF- 1	AA	Default
Reddit	NETMF- 2	AA	AA	Default
Epinions	NETMF- 1	AA	AA	Default
Facebook2	NETMF- 2	AA	AA	Default
Digg	NETMF- 2	NETMF- 1	NETMF- 1	Default
Protein-Soy	NETMF- 1	NETMF- 2	NETMF- 2	Default

References

1. Adamic LA, Adar E (2003) Friends and neighbors on the web. Soc Netw 25(3):211–230
2. Alivisatos AP, Chun M, Church GM, Greenspan RJ, Roukes ML, Yuste R (2012) The brain activity map project and the challenge of functional connectomics. Neuron 74(6):970–974
3. Bawa M, Condie T, Ganesan P (2005) Lsh forest: self-tuning indexes for similarity search. In WWW, pp 651–660
4. Belth C, Büyükçakır A, Koutra D (2020) A hidden challenge of link prediction: Which pairs to check? In: ICDM, pp 831–840. IEEE
5. Belth C, Zheng X, Koutra D (2020) Mining persistent activity in continually evolving networks. In: KDD
6. Charikar MS (2002) Similarity estimation techniques from rounding algorithms. In: STOC
7. Donnat C, Zitnik M, Hallac D, Leskovec J (2018) Learning structural node embeddings via diffusion wavelets. In: KDD, pp 1320–1329. ACM
8. Duan L, Ma S, Aggarwal C, Ma T, Huai J (2017) An ensemble approach to link prediction. In: IEEE TKDE 29(11)
9. Gao M, Chen L, He X, Aoying Z (2018) Bipartite network embedding. In: SIGIR, Bine
10. Grover A, Leskovec J (2016) node2vec: Scalable feature learning for networks. In: KDD, pp 855–864. ACM
11. Hamilton WL, Ying R, Leskovec J (2017) Representation learning on graphs: methods and applications. IEEE Data Eng Bull 40(3):52–74
12. Heimann M, Shen H, Safavi T, Danai K (2018) Representation learning-based graph alignment. In: CIKM, REGAL
13. Di Jin, Heimann M, Safavi, T Wang M, Lee W, Snider L, Koutra D (2019) Smart roles: inferring professional roles in email networks. In: KDD, pp 2923–2933. ACM
14. Joshi U, Urbani J (2020)Searching for embeddings in a haystack: link prediction on knowledge graphs with subgraph pruning. In: WebConf
15. Kipf TN, Welling M (2016) Variational graph auto-encoders. In: NeurIPS workshop on Bayesian deep learning
16. Kunegis J (2013) Konect: the koblenz network collection. In: WWW
17. Latouche P, Birmelé E, Ambroise C et al (2011) Overlapping stochastic block models with application to the French political blogosphere. Ann Appl Stat 5(1):309–336
18. Leskovec J, Krevl A (2014) SNAP datasets: stanford large network dataset collection. http://snap.stanford.edu/data
19. Levin DA, Peres Y (2017) Markov chains and mixing times, volume 107. American Mathematical Soc
20. Liben-Nowell D, Kleinberg J (2007) The link-prediction problem for social networks. ASIS&T 58(7):1019–1031
21. Martínez V, Berzal F, Cubero J-C (2016) A survey of link prediction in complex networks. CSUR 49(4):1–33
22. Mehta N, Carin L, Rai P (2019) Stochastic blockmodels meet graph neural networks. In: ICML
23. Miller K, Michael IJ, Thomas LG (2009) Nonparametric latent feature models for link prediction. In: NeurIPS
24. Newman MEJ (2003) Mixing patterns in networks. Phys Rev E 67(2)
25. Nowicki K, Snijders TAB (2001) Estimation and prediction for stochastic blockstructures. ASIS&T 96(455):1077–1087
26. Pachev B, Webb B (2018) Fast link prediction for large networks using spectral embedding. J Complex Netw 6(1):79–94
27. Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. In: KDD, pp 701–710. ACM
28. Qiu J, Dong Y, Ma H, Li J, Wang K, Tang J (2018) Network embedding as matrix factorization: unifying deepwalk, line, pte, and node2vec. In: WSDM, pp 459–467. ACM
29. Ribeiro LFR, Saverese PHP, Figueiredo DR (2017) struc2vec: learning node representations from structural identity. In: KDD, pp 385–394. ACM
30. Rossi R, Ahmed N (2015) The network data repository with interactive graph analytics and visualization. In: AAAI
31. Rossi RA, Di J, Kim S, Ahmed S, Koutra D, Lee JB (2020) On proximity and structural role-based embeddings in networks: Misconceptions, techniques, and applications. TKDD
32. Safavi T, Koutra D, Meij E (2020) Evaluating the calibration of knowledge graph embeddings for trustworthy link prediction. In: EMNLP
33. Song D, Meyer DA, Tao D (2015) Top-k link recommendation in social networks. In: ICDM, pp 389–398. IEEE

34. Sporns O, Tononi G, Kötter R (2005) The human connectome: a structural description of the human brain. PLoS Comput Biol 1(4):e42
35. Tang J, Qu M, Mei Q (2015) Pte: predictive text embedding through large-scale heterogeneous text networks. In: KDD, pp 1165–1174. ACM
36. Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q (2015) Line: large-scale information network embedding. In: WWW, pp 1067–1077. ACM
37. Tsybakov AB (2008) Introduction to nonparametric estimation. Springer Science & Business Media, Berlin
38. Varshney LR, Chen BL, Paniagua E, Hall DH, Chklovskii DB (2011) Structural properties of the caenorhabditis elegans neuronal network. PLoS Comput Biol 7(2):e1001066
39. Wang J, Shen HT, Song J, Ji J (2014) Hashing for similarity search: a survey. arXiv preprint arXiv:1408.2927
40. Zhang M, Chen Y (2018) Link prediction based on graph neural networks. In: NeurIPS, pp 5165–5175
41. Zhu J, Xingyu L, Heimann M, Koutra D (2021) Node proximity is all you need: Unified structural and positional node and graph embedding. In: SDM, SIAM



Caleb Belth is a PhD candidate in Computer Science and Engineering at the University of Michigan. His research has contributed methods for choosing unlinked pairs of nodes to investigate further with a link prediction method, identifying subtle but persistent patterns in edge streams, and for discovering errors and missing information in incomplete knowledge graphs. Applications of this work include anomaly detection, suspicious behavior discovery, and city/urban planning. He has received an NSF Graduate Research Fellowship, a National Defense Science and Engineering Graduate (NDSEG) fellowship (declined), and a Richard F. and Eleanor A. Towner Prize for Distinguished Academic Achievement. He received a B.S. in Computer Science from Purdue University.

Alican Büyükçakır is a Software Engineer at Token Financial Technologies, Turkey. Previously, he spent two years as a research assistant in the Computer Science and Engineering department at the University of Michigan, under GEMS Lab, where his research focused on anomaly detection in temporal graphs and graph adversarial attacks. He received his B.S. and M.S. in Computer Science from Bilkent University, Turkey.

Danai Koutra is an Associate Director of the Michigan Institute for Data Science (MIDAS) and an Associate Professor in Computer Science and Engineering at the University of Michigan, where she leads the Graph Exploration and Mining at Scale (GEMS) Lab. Her research focuses on principled, practical, and scalable methods for large-scale real networks, and her interests include graph summarization, graph representation learning, graph neural networks, knowledge graph mining, graph learning, similarity and alignment, and anomaly detection. She has won an NSF CAREER award, an ARO Young Investigator award, the 2020 SIGKDD Rising Star Award, research faculty awards from Google, Amazon, Facebook and Adobe, a Precision Health Investigator award, the 2016 ACM SIGKDD Dissertation award, and an honorable mention for the SCS Doctoral Dissertation Award (CMU). She has eight award-winning papers at top data mining conferences. She is an Associate Editor of ACM TKDD, and has served multiple times in the organizing committees of all the major data mining conferences. She has worked at IBM, Microsoft Research, and Technicolor Research. She earned her Ph.D. and M.S. in Computer Science from CMU, and her diploma in Electrical and Computer Engineering at the National Technical University of Athens.