NODE2BITS: Compact Time- and Attribute-aware Node Representations for User Stitching

Di Jin[1] (✉), Mark Heimann[1], Ryan A. Rossi[2], and Danai Koutra[1]

[1] University of Michigan, Ann Arbor
[2] Adobe Research

Abstract. Identity stitching, the task of identifying and matching various online references (*e.g.*, sessions over different devices and timespans) to the same user in real-world web services, is crucial for personalization and recommendations. However, traditional user stitching approaches, such as grouping or blocking, require quadratic pairwise comparisons between a massive number of user activities, thus posing both computational and storage challenges. Recent works, which are often application-specific, heuristically seek to reduce the amount of comparisons, but they suffer from low precision and recall. To solve the problem in an application-independent way, we take a heterogeneous network-based approach in which users (nodes) interact with content (*e.g.*, sessions, websites), and may have attributes (*e.g.*, location). We propose NODE2BITS, an efficient framework that represents multi-dimensional features of node contexts with binary hashcodes. NODE2BITS leverages *feature-based temporal* walks to encapsulate short- and long-term interactions between nodes in heterogeneous web networks, and adopts SimHash to obtain compact, binary representations and avoid the quadratic complexity for similarity search. Extensive experiments on large-scale real networks show that NODE2BITS outperforms traditional techniques and existing works that generate real-valued embeddings by up to 5.16% in $F1$ score on user stitching, while taking only up to 1.56% as much storage.

1 Introduction

Personalization and recommendations increase user satisfaction by providing relevant experiences and handling the online information overload in news, web search, entertainment, and more [13,25]. Accurately modeling user behavior and preferences over time are at the core of personalization. However, tracking user activity online is challenging as users interact with tens of internet-enabled devices from different locations daily, leading to fragmented user profiles. Without unified profiles, the observed user data are sparse, non-representative of the population, and insufficient for accurate predictions that drive business success.

In this work, we tackle the problem of *identity or user stitching*, which aims to identify and group together logged-in and anonymous sessions that correspond to the *same* user despite taking place across different channels, platforms, devices and browsers [28]. This problem is a form of entity or identity resolution [14,2],

also known as entity linking, record linkage, and duplicate detection [6,20,2]. Unlike entity resolution where textual information per user (*e.g.*, name, address) is available, identity stitching relies solely on user interactions with online content and web metadata. Although cookies can help stitch several different sessions of the same user, many users have multiple cookies (*e.g.*, a cookie for each device or web browser) [8], most cookies expire after a short time, and therefore cannot help to stitch users over time. Similarly, IP addresses change across locations resulting in fragmentation or even erroneous stitching between users who have the same IP address at different times (e.g., airports). Meanwhile, fingerprinting approaches [12] capture user similarity based on device or browser configurations, not on behavioral patterns that remain consistent across devices or browsers. On the other hand, exhaustive solutions for entity resolution require quadratic number of comparisons between all pairs of entities, which is computationally intractable for large-scale web services. This can be partially handled via the heuristic of blocking [22], which groups similar entity descriptions into blocks, and only compares entities within the same block.

To overcome these challenges and better tailor to the user stitching setup, our solution is based on the idea that the same user accesses *similar content* across platforms and has *similar behavior* over time. We model the user interactions with different content and platforms over time in a dynamic heterogeneous network, where user stitching maps to the identification of *nodes* that correspond to the same real-world entity. Motivated by the success of node representation learning, we aim to find embeddings of time-evolving 'user profiles' over this rich network of interactions. For large-scale graphs, however, the customary dense node representations for each node can often impose a formidable memory requirement, on par with that of the original (sparse) adjacency matrices [1]. Thus, to efficiently find *sparse* binary representations and link entities based on similar activity while avoiding the pairwise comparison of all user profiles, we solve the following problem:

Problem 1 (Temporal, Hash-based Node Embeddings)**.** *Given a graph $G(V, E)$, the goal of hash-based network embedding is to learn a function $\chi : V \to \{0, 1\}^d$ such that the derived* binary *d-dimensional embeddings (1) preserve similarities in interactions in G, (2) are space-efficient, and (3) accurately capture temporal information and the heterogeneity of the underlying network.*

We introduce a *general* framework called NODE2BITS that captures temporally-valid interactions between nodes in a network, and constructs the contexts based on topological features and (optional) side information of entities involved in the in-

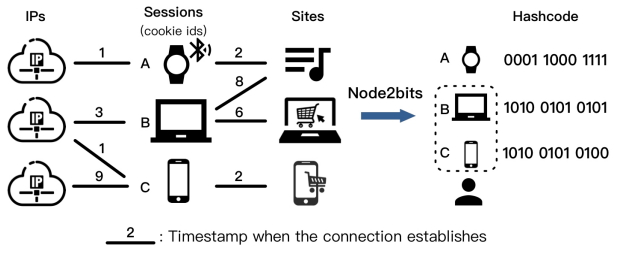

Fig. 1: NODE2BITS overview. NODE2BITS encodes the temporal, heterogeneous information into binary hashcode for efficient user stitching.

teraction. These feature-based contexts are then turned into histograms that incorporate node type information at different *temporal distances*, and are mapped to binary hashcodes through SimHash [5]. Thanks to locality sensitive hashing [32], the hashcodes, which are time-, attribute- and structure-aware, preserve the similarities in temporal interaction patterns in the network, and achieve both space and computational efficiency for similarity search. Given these sparse, hash-based embeddings of all entities, we then cast user stitching as a supervised binary classification task or a hashing-based unsupervised task. As an example shown in Fig. 1, devices B and C are associated with identical IPs and similar online sales websites visited afterwards, thus they are encoded similarly and could correspond to the same user. Our contributions are:

- **Embedding-based Formulation:** Going beyond traditional blocking techniques, we formulate the problem of user stitching as the problem of finding temporal, hash-based embeddings in heterogeneous networks such that they maintain similarities between user interactions over time.
- **Space-efficient Embeddings:** We propose NODE2BITS, a practical, intuitive, and fast framework that generates compact, binary embeddings suitable for user stitching. Our method combines random walk-based sampling of contexts, their feature-based histogram representations, and locality sensitive hashing to preserve the heterogeneous equivalency of contexts over time.
- **Extensive Empirical Analysis:** Our experiments on real-world networks show that NODE2BITS outputs a space-efficient binary representation which uses between 63× and 339× less space than the baselines while achieving comparable or better performance in user stitching tasks. Moreover, NODE2BITS is scalable for large real-world temporal and heterogeneous networks.

2 Preliminaries and Definitions

Before we introduce NODE2BITS, we discuss two key concepts that our method is based on: dynamic heterogeneous network model, and temporal random walks. We give the main symbols and their definitions in Table 1.

2.1 Dynamic Heterogeneous Network Model

As we mentioned above, we model the user interactions with content, websites, devices etc. as a heterogeneous network, which is formally defined as:

Definition 1 (HETEROGENEOUS NEWORK)**.** *A heterogeneous network* $G = (V, E, \psi, \xi)$ *is comprised of (i) a nodeset* V *and edgeset* E*, (ii) a mapping* $\psi : V \to \mathcal{T}_V$ *of nodes to their types, and (iii) a mapping* $\xi : E \to \mathcal{T}_E$ *to edge types.*

Many graph types are special cases of heterogeneous networks: (**1**) homogeneous graphs have $|\mathcal{T}_V| = |\mathcal{T}_E| = 1$ type; (**2**) k-partite graphs consist of $|\mathcal{T}_V| = k$ and $|\mathcal{T}_E| = k - 1$ types; (**3**) signed networks have $|\mathcal{T}_V| = 1$ and $|\mathcal{T}_E| = 2$ types; and (**4**) labeled graphs have a single label per node/edge.

Table 1: Summary of major symbols and their definitions.

Symbol	Definition
$G(V, E, \xi, \psi)$	(un)directed and (un)weighted heterogeneous network with nodeset V, edgeset E, a mapping ξ from nodes to node types, and an edge mapping ψ, resp.
$\|V\| = N, \|E\| = M$	number of nodes and edges in G
$\mathcal{T}_V, \|\mathcal{T}_V\|$; $\mathcal{T}_E, \|\mathcal{T}_E\|$	set of node/edge types in the heterogeneous graph and its size, resp.
$\mathbf{F}$	$N \times \|\mathcal{F}\|$ feature matrix including node attributes and derived features
f_{ij}, $f_{(j)}$	$(i,j)^{th}$ element of $\mathbf{F}$ and index of its j^{th} feature, resp.
$\mathcal{W}$	set of random walks
$(\mathbf{w}_L)_{L\in\mathbb{N}}, \mathbf{w}_L[u]$	sequence of nodes in a random walk of length L, and the index of node u, resp.
L	the maximum length of a random walk
Δt	'temporal distance' in $\mathcal{W}$ based on temporally ordered edge transitions
$\mathcal{C}_u^{\Delta t}$, $\mathcal{C}_u^{\Delta t}\|f$	context of node u at distance Δt, and feature-based context, resp.
$g_i : \mathcal{C} \to \{0,1\}$	i^{th} LSH function that hashes a node context into a binary value
$K^{\Delta t}$, K	embedding dimension at distance Δt, and output dimension $K = \sum_{\Delta t=1}^{\text{MAX}} K^{\Delta t}$
$\mathbf{h}(\mathcal{S}), \mathbf{h}(\mathcal{S}\|\cdot)$	unconditional and conditional b-bin histogram of values in enclosed set $\mathcal{S}$, resp.
$\mathbf{Z}$	$N \times K$ output binary embeddings or hashcodes

Most real networks capture evolving processes (e.g., communication, browsing activity) and thus change over time. Instead of approximating a dynamic network as a sequence of lossy discrete static snapshots $G_1, \ldots, G_T$, we model the *temporal interactions* in a *lossless* fashion as a *continuous-time dynamic network* [21].

Definition 2 (Continuous-Time Dynamic Network)**.** *A continuous-time dynamic, heterogeneous network $G = (V, E_\tau, \psi, \xi, \tau)$ is a heterogeneous network with E_τ temporal edges between vertices V, where $\tau : E \to \mathbb{R}^+$ is a function that maps each edge to a corresponding timestamp.*

2.2 Temporal Random Walks

A walk on a graph is a sequence of nodes where each pair of successive nodes are connected by an edge. Popular network embedding methods generate walks using randomized procedures [23,15] to construct a corpus of node IDs or node context. In continuous-time dynamic networks, a *temporally valid* walk is defined

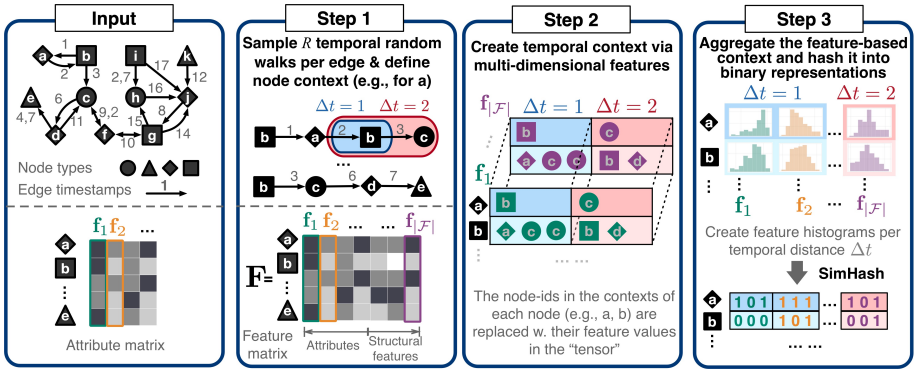

Fig. 2: NODE2BITS workflow. Given a graph and its attribute matrix (optional), NODE2BITS (1) samples *temporal* random walks to obtain sequences that respect time; derives contexts at different temporal distances, as well as the feature matrix $\mathbf{F}$ (temporal contexts of a and b are derived from the walk $\{b, a, b, c\}$ (2) creates temporal contexts based on multi-dimensional features in $\mathbf{F}$, and (3) aggregates them into feature-based histograms to obtain sparse, binary, similarity-preserving embeddings via SimHash.

as a sequence of nodes connected by edges with non-decreasing timestamps (*e.g.*, representing the order that user-content interactions occurred) and they were first used for embeddings in [21].

Definition 3 (TEMPORAL WALK)**.** *A temporal walk of length L from v_1 to v_L in graph $G = (V, E, \psi, \xi)$ is a sequence of vertices $\langle v_1, v_2, \cdots, v_L \rangle$ such that $\langle v_i, v_{i+1} \rangle \in E_\tau$ for $1 \leq i < L$, and the timestamps are in valid temporal order: $\tau(v_i, v_{i+1}) \leq \tau(v_{i+1}, v_{i+2})$ for $1 \leq i < (L-1)$.*

3 NODE2BITS: Hash-based Emdedding Framework

Motivated by the task of user stitching, we aim to develop NODE2BITS to compactly describe each node/entity in the context of *realistic interactions* (Problem 1). The requirements for NODE2BITS naturally follow from the user stitching application. It should **(R1)** support heterogeneous networks where the nodes and edges can be of any arbitrary type (*e.g.*, a user, web page, IP, tag, spatial location); **(R2)** preserve the temporal validity of the events and interactions in the data; **(R3)** scale in runtime to large networks with millions of nodes/edges; and **(R4)** scale in memory requirements with space-efficient yet powerful binary embeddings that capture ID-independent similarities. Next we detail the three main steps of NODE2BITS: (§ 3.1) Sampling temporal random walks and defining temporal contexts; (§ 3.2) Constructing temporal contexts based on multi-dimensional features; (§ 3.3) Aggregating and hashing contexts into sparse embeddings. We give the overview of NODE2BITS in Figure 2 and Algorithm 1. For reproducibility, the code is available at `https://github.com/GemsLab/node2bits`.

3.1 Temporal Random Walk Sampling

The first step of NODE2BITS is to captures node interactions in its context, which is important for the user stitching task: instead of simple interactions corresponding to pairwise edges, it samples more complex interaction sequences via random walks. But unlike many existing representation learning approaches [23,15], our method samples *realistic* interactions in the order that they happen via L-step *temporal* random walks (Definition 3 [21]), thus satisfying requirement **R2**.

NODE2BITS defines the *temporal context* $\mathcal{C}_u^{\Delta t}$ of node u at temporal distance Δt as the *collection of entities* that are at Δt-hops away from node u in the sampled random walks. Formally:

$$\mathcal{C}_u^{\Delta t} = \{v \ : \ |\mathbf{w}_L[v] - \mathbf{w}_L[u]| = \Delta t, \ \forall \mathbf{w}_L \in \mathcal{W}\}, \tag{1}$$

where $\mathbf{w}_L[\cdot]$ is the index of the corresponding node in the random walk $(\mathbf{w}_L)_{L\in\mathbb{N}}$. For example, in Figure 2 (Step 1) the context of node a at temporal distance 2 is $\mathcal{C}_a^{\Delta t=2} = \{c\}$ (highlighted in red). Depending on the temporal context that we want to capture, the Δt can vary up to a MAX distance. Intuitively, small values of temporal distance capture more *direct* interactions and similarities between entities. In static graphs, Δt simply corresponds to the distance between nodes in the sampled sequences, without capturing any temporal information.

Temporal locality. The context that is defined above does not explicitly incorporate the *time elapsed* between consecutively sampled interactions. However, when modeling temporal user interactions, it is important to distinguish between short-term and long-term transitions. Inspired by [21], NODE2BITS accounts for the *closeness* or *locality* between consecutive contexts (*i.e.*, $\mathcal{C}_u^{\Delta t}$ and $\mathcal{C}_u^{\Delta t+1}$) through different biased temporal walk policies. For example, in the short-term policy, the transition probability from node u to v is given as the softmax function:

$$P[v|u] = \frac{\exp\left(-\tau(u,v)/d\right)}{\sum_{i\in\Gamma_\tau(u)} \exp\left(-\tau(u,i)/d\right)} \tag{2}$$

where $\tau()$ maps an edge to its timestamp, $d = \max_{e\in E_\tau} \tau(e) - \min_{e\in E_\tau} \tau(e)$ is the total duration of all timestamps, and $\Gamma_\tau(u)$ is the set of temporal neighbors reached from node u through temporally valid edges. Similarly, in the long-term policy, the transition probability from node u to v is given as in Equation (2) but with positive signs in the numerator and denominator.

3.2 Temporal Context based on Multi-dimensional Features

The context in Eq. (1) depends on the node identities (IDs). However, in a multi-platform environment, a single entity may have multiple node IDs, thus contributing to seemingly different contexts. To generate ID-independent contexts that are appropriate for user stitching, we make the temporal contexts attribute- or feature-aware (**R1**), by building on the assumption that corresponding or similar entities have similar features. Formally, we assume that a network may have a set of input node attributes (*e.g.*, IP address, device type), as well as

a set of derived topological features (*e.g.*, degree, PageRank), all of which are stored in a $N \times |\mathcal{F}|$ feature matrix $\mathbf{F}$ (Figure 2, Step 1). We then generalize our random walks to not only respect time (**R2**) [21], but also capture this feature information using the notion of attributed/feature-based walks proposed in [1]:

Definition 4 (Feature-based Temporal Walk)**.** *A* feature-based temporal walk *of length L from node v_1 to v_L in graph G is defined as a sequence of feature values corresponding to the sequence of vertices in a valid temporal walk (Dfn. 3). For the j^{th} feature $f_{(j)}$, the corresponding feature-based temporal walk is*

$$\langle w_{L,f_{(j)}} \rangle_{L \in \mathbb{N}} = \langle f_{v_1,j}, f_{v_2,j}, \ldots, f_{v_L,j} \rangle, \tag{3}$$

where $f_{v_i,j}$ is the value of the j^{th} feature for node v_i, stored in matrix $\mathbf{F}$.

Our definition is general as it allows walks to obey time while allowing each node to have a d-dimensional vector of input attribute values and/or derived structural features, which can be discrete or real-valued [1].

Temporally-valid, multi-dimensional feature contexts. node2bits extends the previously generated temporal contexts to incorporate node features and remove the dependency on node IDs. Following the definition of feature-based temporal walks, given $|\mathcal{F}|$ features, our method generates $|\mathcal{F}|$-dimensional contexts per node u and temporal distance Δt by replacing the node IDs in Equation (1) with their corresponding feature values (Figure 2, Step 2). Formally, the temporally-valid, multi-dimensional feature contexts are defined as:

$$\mathcal{C}_u^{\Delta t} | f_{(j)} = \{ f_{v,j} \ : \ \forall v \in \mathcal{C}_u^{\Delta t} \} \ \ \forall \text{ feature } f_{(j)} \in \mathcal{F}, \tag{4}$$

where $f_{v,j}$ is the value of the j^{th} feature for node v.

3.3 Feature-based Context Aggregation and Hashing

The key insight in user stitching is that each user interacts with similarly 'typed' entities through similar relations over time: for example, in online-sales logs, a user likely browses similar types of goods in logged-in and anonymous sessions; and in online social networks, accounts sharing near-identical interaction patterns, such as replies or shares, are potentially from the same person. Based on this insight, node2bits augments the previously generated temporal, multi-dimensional feature contexts with node types (and implicitly the corresponding relations or edge types), which is a key property of heterogeneous networks (**R1**). It subsequently aggregates them and derives similarity-preserving and space-efficient, binary entity representations (**R4**) via locality sensitive hashing.

Context Aggregation. Unlike existing works that aggregate contextual features into a single value such as mean or maximum [16,27], node2bits aggregates them into *less* lossy representations: *histograms* tailored to heterogeneous networks by

distinguishing between node types (**R1**). Specifically, it models the transitional dependency across node and relation types by further conditioning the derived contexts in Equation (4) on the node types $p_i \in \mathcal{T}_V$ (*i.e.*, each temporal context consists of the features of only one node type). We denote the temporal contexts conditioned on both features and node types as $\mathcal{C}_u^{\Delta t}|f, p$. The final histogram representation of node u at temporal distance Δt consists of the concatenation of the histograms over the conditional contexts at Δt (Figure 2, Step 3):

$$\mathbf{h}(\mathcal{C}_u^{\Delta t}) = [\mathbf{h}(\mathcal{C}_u^{\Delta t} \mid f_{(1)}, p_1), \mathbf{h}(\mathcal{C}_u^{\Delta t} \mid f_{(2)}, p_1), \ldots, \mathbf{h}(\mathcal{C}_u^{\Delta t} \mid f_{(|\mathcal{F}|)}, p_{|\mathcal{T}_V|})]. \quad (5)$$

In this representation, the features are binned logarithmically to account for the often skewed distributions of structural features. We note that the histograms can be further extended to edge types as well, for example by distinguishing nodes that are connected by different types of edges.

Similarity-preserving Representations via Hashing. Locality sensitive hashing (LSH) has been widely used for searching nearest neighbors in large-scale data mining [32]. In this work, we adopt SimHash [5] to obtain similarity-preserving and space-efficient representations (**R4**) for all the entities in the heterogeneous network based on their aggregated time-, attribute-, and node type-aware contexts (Equation 5).

Given a node-specific histogram $\mathbf{h}(\mathcal{C}_u^{\Delta t}) \in \mathbb{R}^d$ (with dimensionality $d = |\mathcal{F}||\mathcal{T}_V| \cdot b$, and b being the number of logarithmic bins for the features), SimHash generates a $K^{\Delta t}$-dimensional[3] binary hashcode or sketch $\mathbf{z}_u^{\Delta t}$ by projecting the histogram to $K^{\Delta t}$ random hyperplanes $\mathbf{r}_i \in \mathbb{R}^d$ as follows:

$$g_i(\ \mathbf{h}(\mathcal{C}_u^{\Delta t})\ ) = \text{sign}\left(\mathbf{h}(\mathcal{C}_u^{\Delta t}) \cdot \mathbf{r}_i\right) \quad (6)$$

In practice, the hyperplanes do not need to be chosen uniformly at random from a multivariate normal distribution, but it suffices to choose them uniformly from $\{-1, 1\}^d$. The important property of locality sensitive hashing that guarantees that the similarities in the histogram space (which captures the temporal interactions between entities in G) are maintained is the following: for the SimHash family, the probability that a hash function agrees for two different vectors is equal to their cosine similarity. More formally, for two nodes u and v:

$$P(\ g_i(\mathbf{h}(\mathcal{C}_u^{\Delta t})) = g_i(\mathbf{h}(\mathcal{C}_v^{\Delta t}))\ ) = 1 - \frac{\cos^{-1} \frac{\mathbf{h}(\mathcal{C}_u^{\Delta t}) \cdot \mathbf{h}(\mathcal{C}_v^{\Delta t})}{|\mathbf{h}(\mathcal{C}_u^{\Delta t})||\mathbf{h}(\mathcal{C}_v^{\Delta t})|}}{180}. \quad (7)$$

In other words, the cosine similarity between nodes u and v in the context-space is projected to the sketch-space and can be measured by the cardinality of matching between $\mathbf{z}_u^{\Delta t}$ and $\mathbf{z}_v^{\Delta t}$, where $\mathbf{z}_\cdot^{\Delta t} = [g_1(\ \mathbf{h}(\mathcal{C}_\cdot^{\Delta t})\ ), g_2(\ \mathbf{h}(\mathcal{C}_\cdot^{\Delta t}), \ldots, g_{K^{\Delta t}}(\ \mathbf{h}(\mathcal{C}_\cdot^{\Delta t})]$.

For each node u in G, the final binary representation is obtained by concatenating the hashcodes for contexts at different temporal distances Δt, resulting in a K-dimensional vector (since $K = \sum_{\Delta t=1}^{\text{MAX}} K^{\Delta t}$):

$$\mathbf{z}_u = [\mathbf{z}_u^{\Delta t=1}\ \mathbf{z}_u^{\Delta t=2}\ \ldots\ \mathbf{z}_u^{\Delta t=\text{MAX}}] \quad (8)$$

[3] We assume that the length of each sketch at distance Δt is given as $K^{\Delta t} = \frac{K}{\text{MAX}}$.

Algorithm 1 NODE2BITS Framework

Require: (un)directed heterogeneous graph $G(V, E, \psi, \xi)$, # random walks R per edge, max walk length L, max temporal distance MAX, embedding dimensionality $K^{\Delta t}$ at dist. Δt

1: For each edge e, perform R temporal walks based on the short- or long-term policy (§ 3.1)
2: Obtain temporal contexts $\mathcal{C}_u^{\Delta t}$ for each node u at temporal distances $\Delta t \leq$ MAX via Eq. (1)
3: Construct feature matrix $\mathbf{F}$ with node attributes (if avail.) and topological features (§ 3.2)
4: Derive feature-based temporal contexts $\mathcal{C}_u^{\Delta t}|f_{(j)}$ by replacing $v \in \mathcal{C}_u^{\Delta t}$ with the feature value $f_{v,j}$, as shown in Eq. (4)
5: **for each** temporal distance $\Delta t = 1,\ldots,$MAX and node $u \in V$ **do**
6: | Obtain u's final histogram $\mathbf{h}(\mathcal{C}_u^{\Delta t})$ over its contexts using Eq. (5)
7: | Obtain a $K^{\Delta t}$-dim, sparse, binary hashcode $z_u^{\Delta t}$ based on (modified) SimHash (§ 3.3)
8: Obtain the binary N2B embeddings $\mathbf{z}_u$ of all nodes across temporal distances Δt via Eq. (8)
9: Perform (un)supervised user stitching via binary classification or hashing (§ 4.1,4.3)

where we replace the -1 bits with 0s to achieve a more space-efficient representation (**R4**). An example is shown in the second half of Step 3 in Figure 2, where the blue shades denote histograms and sketches for contexts in temporal distance $\Delta t = 1$, and red shades correspond to $\Delta t = 2$. Thus, the K-dimensional representation for each node, $\mathbf{z}_u \in \{0, 1\}^K$, captures the similarities between time-, feature- and node type-aware histograms across multiple temporal distances Δt. The similarity between two nodes' histograms can be quickly estimated as the proportion of common bits in their binary representations $\mathbf{z}_.$.

Given these representations, we can perform user stitching by casting the problem as supervised binary classification or an unsupervised task based on the output of hashing (Eq. (8)), which we discuss in Section 4.1. Putting everything together, we give the pseudocode of NODE2BITS in Algorithm 1 and its detailed version (for reproducibility) in supplementary material A. The runtime computational complexity of NODE2BITS is $\mathcal{O}(MRL + NK)$, which is linear to the number of edges when $M \gg N$ as K is relatively small (**R3**). The output space complexity is $\mathcal{O}(NK)$-bit. NODE2BITS requires even less storage if the binary vectors are represented in the sparse format (see Section 4.4 for empirical results). We provide detailed complexity analysis in supplementary material B.

4 Experiments

We perform extensive experiments on real-world heterogeneous networks to answer the following questions: (**Q1**) Is NODE2BITS effective in the user stitching task, and how does it compare to traditional stitching and embedding methods? (§ 4.2-4.3) (**Q2**) Does NODE2BITS have low space requirements, and is it more space-efficient than the baselines? (§ 4.4) (**Q3**) Is NODE2BITS scalable? (§ 4.5)

4.1 Experimental Setup

We ran our analysis on Mac OS platform with 2.5GHz Intel Core i7, 16GB RAM.

Data. We use five real-world heterogeneous networks from the Network Repository [26], as well as a real, proprietary user stitching dataset, 'Company X' web logs. The latter data form a temporal heterogeneous network consisting of web sessions of user devices and their IP addresses. (Our framework is also capable of modeling domain-specific features, such as user-agent strings and geolocation [19], if this is available. Even without them, however, we achieve strong performance.) High degree nodes representing anomalous behavior (*e.g.*, bots or public WiFi hotspots) are filtered out. We give the statistics of all the networks in Table 2, and additional details in Appendix C.

Table 2: Network statistics and properties for our six real-world datasets. 'D': directed; 'W': weighted; 'H': heterogeneous; 'T': temporal network.

Data	Nodes	Edges	$\lvert T_V \rvert$	D	W	H	T
citeseer	4460	2892	2		✓	✓	
yahoo	100,058	1,057,050	2	✓	✓	✓	
bitcoin	3,783	24,186	1	✓	✓		✓
digg	283,183	6,473,708	2			✓	✓
wiki	1,140,149	7,833,140	1	✓			✓
comp-X	5,500,802	5,291,270	2	✓	✓	✓	✓

Task Setup. With the exception of Section 4.3, we cast the user stitching task as a binary classification problem, where for each pair of nodes we aim to predict whether they correspond to the same entity (*i.e.*, we should stitch them). We use logistic regression with regularization strength 1.0 and stopping criterion 10^{-4}.

For the real user stitching data, we use the held-out, ground-truth information to evaluate our method. For the five real-world networks without known user pairs, we introduce user replicas following a similar procedure to [2]: we sample 5% of the nodes with degrees larger than average, introduce a replica u' for each sampled node u, and distribute the original edges between u and u'. Specifically, each edge remains connected to u with probability p_1, otherwise it connects to the replica node u'. Additionally, each edge that is incident to u has probability p_2 to also connect to u'. Unless otherwise specified, we use $p_1 = 0.6$ and $p_2 = 0.3$.

Given the positive replica pairs, we sample an equal number of negative pairs uniformly at random and include these in the train and test sets. Comp-X, the dataset with ground-truth replicas, also has pre-defined approximately 50/50 train-test splits that we use. Afterwards, embeddings are derived for each node pair by concatenation: $[\mathbf{z}(u), \mathbf{z}(u')]$. Using these node pair embeddings, we learn a logistic regression (LR) model and use it to predict the node pairs that should be stitched in the held-out test set. These are the nodes that correspond to the same entity. We measure the predictive performance of all the methods in terms of AUC, accuracy and $F1$ score.

Baselines. We compare to various methods that target different network types:

- **Homogeneous networks**: *Static*– **(1)** Spectral embeddings or SE [31], **(2)** LINE [30], **(3)** DeepWalk or DW [23], **(4)** node2vec or n2vec [15], **(5)** struc2vec or s2vec [24], and **(6)** DNGR [4]. *Temporal*– **(7)** CTDNE [21].
- **Heterogeneous networks**: **(8)** Common neighbors (CN) [2], **(9)** metapath2vec or m2vec [11], and **(10)** AspEm [29].

The baselines are configured to achieve the optimal performance, for $K = 128$-dimensional embeddings, according to the respective papers. For reproducibility, we describe the settings in Appendix D.

NODE2BITS Setup & Variants. Similar to the baselines, NODE2BITS performs $R = 10$ walks per edge, with length up to $L = 20$, and we set the max temporal distance MAX = 3. We justify these decisions in Appendix E.2. On the largest dataset, Comp-X, we use a maximum walk length $L = 5$ and temporal distance MAX = 2. While various node attributes can be given as input to NODE2BITS, for consistency we derive and use the total, in-/out-degree of each node in $\mathbf{F}$.

We experiment with different variants of NODE2BITS (or N2B for short): (1) **NODE2BITS-0** applies to static networks; (2) **NODE2BITS-SH** uses the short-term tactic in the random walks (§ 3.1); (3) **NODE2BITS-LN** uses the long-term tactic; and (4) **NODE2BITS-U** targets unsupervised user stitching, which most baselines cannot handle (except for CN). To explore our method's performance in unsupervised settings (§ 4.3), we directly 'cluster' the LSH-based, binary node representations $\mathbf{z}_u$ generated by NODE2BITS-0. The idea is that nodes that hash to the same 'bucket' likely map to the same entity and should be stitched. To map entities to buckets we use the banding technique [32]: per band—one per representation $\mathbf{z}^{\Delta t}$ at temporal distance Δt—we apply AND-construction on the output of bit sampling, and then OR-construction across the bands.

4.2 Accuracy in Supervised User Stitching

We start by evaluating the predictive performance of NODE2BITS for supervised user stitching on both static and temporal networks.

Table 3: Entity resolution results for *static* networks. Our method outperforms all the baselines. *OOT = Out Of Time (6 hours), OOM = Out Of Memory (16GB).

	Metric	CN	SE	LINE	DW	n2vec	s2vec	DNGR	m2vec	As-pEm	N2B-0
citeseer	**AUC**	0.9141	0.4846	0.5481	0.5614	0.6188	0.9344	0.5015	0.5546	0.5049	0.9480
	ACC	0.9141	0.5045	0.5372	0.5579	0.6211	0.8936	0.4688	0.5357	0.5223	0.9196
	F1	0.9137	0.5028	0.5371	0.5547	0.6159	0.8926	0.4682	0.5348	0.5222	0.9192
yahoo	**AUC**	0.6851	0.5378	0.8050	0.7640	0.7636	OOT	OOM	OOT	0.4938	0.8088
	ACC	0.6851	0.4760	0.7771	0.7117	0.7233				0.5018	0.8010
	F1	0.6505	0.4375	0.7764	0.7117	0.7231				0.5018	0.7987

Static Networks. In this experiment, we evaluate the effectiveness of introducing multi-dimensional feature contexts. Since static networks lack temporal information, NODE2BITS performs random walks similarly to existing works to collect nodes in structural contexts. The main difference lies in representing diverse feature histograms. We run NODE2BITS against both homogeneous and heterogeneous baselines as shown in Table 3, and observe that it achieves the best performance in all evaluation metrics on both static graphs. NODE2BITS outperforms existing random-walk based methods as expected: node IDs in the contexts is distorted by the replicas generated, thus feature-based methods should prevail. This is also validated by the results for struc2vec, which captures the equivalency of structural feature sequences in embeddings. LINE achieves promising result on yahoo but not on citeseer, as the latter undirected bipartite graph, node distributions of the 2-order contexts explored by LINE are highly correlated and indistinguishable for stitching. On the contrary, CN (common-neighbors) yields promising result on citeseer but not yahoo. This is likely due to the graph structure, which we explain in more detail in Sec.4.3. We encountered out-of-memory errors for DNGR due to the algorithmic complexity and out-of-time-limit for m2vec, s2vec. Overall, NODE2BITS achieves comparable and slightly better performance in both AUC and $F1$ scores with $0.60\% - 1.46\%$ and $0.60\% - 2.87\%$ improvement, respectively.

CONCLUSION 1. *On static graphs,* NODE2BITS *achieves comparable, and slightly better, performance in AUC and F1 score over baselines in the stitching task.*

Temporal Networks. Table 4 depicts the stitching performance of NODE2BITS using both the short- and long-term tactics against the same set of baselines used in static graphs as well as CTDNE, an embedding framework designed for temporal graphs. We observe that NODE2BITS-SH outperforms NODE2BITS-LN in most cases, which is reasonable because NODE2BITS-LN derives shorter contexts

Table 4: Entity resolution results for *temporal* networks: strong performance for NODE2BITS variants. *OOT = Out Of Time (6 hours), OOM = Out Of Memory (16GB).

	Metric	CN	SE	LINE	DW	n2vec	CTDNE	s2vec	DNGR	N2B-0	N2B-SH	N2B-LN
bitcoin	**AUC**	0.7474	0.5828	0.6071	0.6306	0.6462	0.6987	0.8025	0.5909	0.7584	0.7609	0.7380
	ACC				0.6158				0.5526	0.7211	0.7268	0.6737
	F1	0.7174	0.5842	0.5842	0.6158	0.6158	0.6000	0.7263	0.5525	0.7209	0.7271	0.6735
		0.7001	0.5728	0.5828		0.6157	0.5964	0.7263				
digg	**AUC**	0.6217	0.5171	0.7878	0.7398	0.7445	0.6967	OOT	OOM	0.8185	0.7611	0.7587
	ACC				0.6971					0.7982	0.7418	0.7444
	F1	0.6217	0.5152	0.7694	0.6960	0.7013	0.5915			0.7958	0.7411	0.7433
		0.5585	0.3770	0.7683		0.7003	0.5884					
wiki	**AUC**	0.6997	OOT	0.7854	OOM	OOM	0.7707	OOT	OOM	0.8230	0.8259	0.8214
	ACC									0.7145	0.7510	0.7103
	F1	0.6997		0.7132			0.6488			0.7088	0.7476	0.7067
		0.6699		0.7129			0.6398					
comp-X	**AUC**	0.5970	OOM	0.5000	OOM	OOM	OOM	OOT	OOM	0.8095	0.7496	0.7525
	ACC									0.8414	0.7959	0.7975
	F1	0.5970		0.6757						0.8154	0.7581	0.7606
		0.5189		0.4032								

constrained by temporal-order. We also justify the effectiveness of temporal random walk by comparing it with both NODE2BITS-0 and static baselines where we only make use of the graph structures without specifying edge timestamps. We observe that NODE2BITS-0 is the best-performing method for the digg dataset and Comp-X over the temporal variants of NODE2BITS. The reason behind is that there is a tradeoff in constraining temporal walks to respect time: we more accurately model realistic sequences of events at the cost of restricting the possible context. On these particular temporal graphs, walks may already be limited in length by the bipartite structure, so the latter cost becomes more appreciable. Nevertheless, both static and dynamic versions of NODE2BITS almost always outperform other baselines. In particular, across all datasets, NODE2BITS-SH still outperforms the temporal baseline, CTDNE in all cases, which further demonstrates the effectiveness of multi-feature aggregation.

NODE2BITS variants outperform the static methods in nearly all cases except the bitcoin dataset where NODE2BITS-SH achieves lower AUC than struc2vec but higher ACC and $F1$-score. This is because NODE2BITS loses some information when representing the node contexts as binary vectors comparing with real-value representation. However, we consider this loss mild as NODE2BITS still outperforms all the other static baselines. In addition, struc2vec ran out of time on the larger datasets while NODE2BITS achieves promising performance efficiently

with $3.90\% - 5.16\%$ improvement in AUC and $3.58\% - 4.87\%$ improvement in $F1$ score than the best baseline method. At the same time, our approach uses much less information than the static methods, since the length of the temporal walks are typically shorter than random walks that do not have to respect time.

Conclusion 2. *Dynamic and static variants of* node2bits *outperform the other baselines by up to* 5.2% *in AUC and* 4.9% *in* $F1$ *score. Between two dynamic variants of* node2bits, node2bits-sh *performs better than* node2bits-ln.

Restricting the Output Space Requirements. To evaluate the performance of stitching with explicit storage requirement, we hash the real-value embeddings given by baselines into binary and achieve output storage to be consistent with node2bits. We observe that node2bits still achieves the best performance (refer to Table 6 of Section E.1 from the supplementary materials for more details).

4.3 Accuracy in Unsupervised User Stitching

As mentioned in Section 4.1, node2bits can naturally perform unsupervised user stitching by leveraging the generated node representations as hashcodes. Only nodes mapped to the same 'buckets' are candidates for stitching together. This process allows us to stitch entities without involving quadratic comparisons between all pairs of nodes in the graph. Similarly, CN outputs a set of nodes sharing a certain amount of neighbors as the candidates to be stitched together. We evaluate the quality of hashing given by node2bits-u against CN, and make use of the candidates to predict the testing set of node pairs given by following the same setup in Sec. 4.2 in an unsupervised scheme.

Based on the results in Table 5, we observe that node2bits-u outperforms CN on every dataset other than citeseer. The reason is that in this "author contributes to paper" dataset, author references appearing in the same set of papers have high probability to correspond to the same researcher in reality. Therefore the assumption made by CN suits well this scenario, whereas node2bits hashes nodes with similar features in the context instead of those with similar neighbor identities (IDs). For datasets with less strict cross-type relationship, node2bits achieves $2.81\% - 15.12\%$ improvement in accuracy ACC and $4.96\% - 26.66\%$ improvement in $F1$ score (including digg, another bipartite graph with inner connected components of the same node types).

Conclusion 3. *The unsupervised variant of* node2bits, node2bits-u, *outperforms CN on most graphs.*

4.4 Output Storage Efficiency

Next we evaluate space efficiency of our proposed method over baselines that output node embeddings. Instead of real-value matrices, the binary hashcodes generated by node2bits can be stored in the sparse format so presumably it should take trivial storage. We visualize the storage requirements in Figure 3 and provide detailed storage usage in Table 7 in the supplementary material.

Table 5: Unsupervised stitching performance between CN and NODE2BITS

Metric	citeseer		yahoo		bitcoin		digg		wiki	
	CN	N2B-U	CN	N2B-U	CN	N2B-U	CN	N2B-U	CN	N2B-U
ACC	0.9141	0.8661	0.6851	0.7553	0.7474	0.7684	0.6217	0.7157	0.6997	0.7350
F1	0.9137	0.8660	0.6505	0.7518	0.7301	0.7663	0.5585	0.7074	0.6699	0.7349

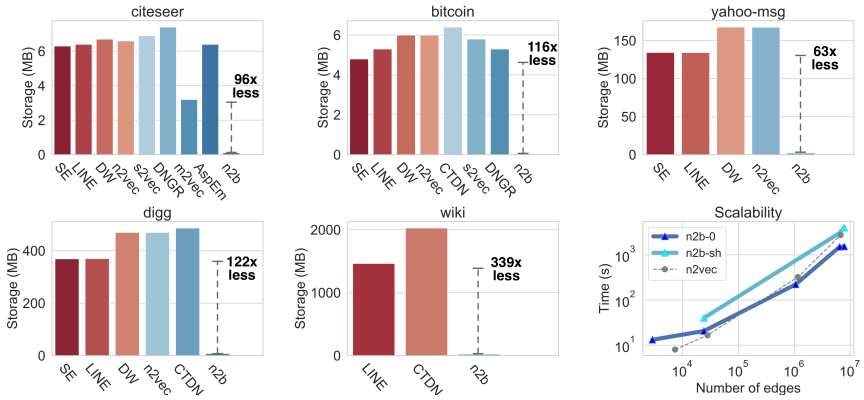

Fig. 3: First 5 plots: output storage in MB for all the methods that completed successfully in five datasets. The approach is also shown to be scalable for large graphs.

CONCLUSION 4. *Compared to the other methods,* NODE2BITS *uses between* 63× *and* 339× *less space (while always achieving comparable or better stitching performance as shown in Section 4.2).*

4.5 Scalability

To evaluate the scalability, we report the runtime of applying NODE2BITS to obtain node representations for the datasets shown in Table 2 versus their numbers of edges. We also visualize the runtime of node2vec as reference, as it is designed for large graphs and is implemented in the same language (Python). Based on the last subplot in Figure 3, we observe that NODE2BITS scales similarly as node2vec with less runtime space as node2vec ran out of memory on the largest dataset (wiki). As shown in Section B, the worst-case time complexity is linear in the edges. We provide the detailed runtime of all methods in Table 8 in the Appendix.

5 Related Work

Entity Resolution. Entity resolution (the general problem under which user stitching falls) has been widely studied and applied in different domains such as

databases and information retrieval [10,14]. Traditional methods that are based on distances can be broadly categorized into (1) pairwise-ER [7], which independently decide which pairs are same entity based on a distance threshold, and (2) clustering [8], which links nodes in the same cluster. However, these methods suffer from high computational cost due to pairwise comparisons, and do not scale to large datasets. Other techniques range from supervised classification [28] to probabilistic soft logic [19] or fingerprinting [12] using side information (*e.g.*, user-agent strings, other web browser features, geo-location). These methods tend to be problem- or even data-specific. By modeling the data with a heterogeneous, dynamic network, we propose a general method that can use both node features (optional) *and* graph structure.

Node embeddings. Node embedding or representation learning methods aim to preserve the notion of node similarity into low-dimensional vector space. Most methods [15,23,30] and the state-of-the-art methods formulated for heterogeneous or dynamic networks [11,21], define node similarity based on co-occurrence or proximity in the original network (Appendix F). However, in the user stitching problem, it is possible that corresponding entities may not connect to the same entities, resulting in lower proximity-based similarity. Embeddings preserving structural iddntity [24,1,17] overcome this drawback even if nodes do not connect to the same entities. Our method also has this property over a variety of graph settings (heterogeneous, dynamic), at greater space efficiency thanks to hashing.

Locality sensitivity hashing (LSH). LSH was first introduced as a randomized hashing framework for efficient approximate nearest neighbor (ANN) search in high dimensional space [18]. It specifies a family of hash functions, $\mathcal{H}$, that maps similar items to the same bucket identified through hash codes with higher probability than dissimilar items [32]. Variants of LSH families for different distances have been widely studied, such as SimHash for cosine distance [5], min-hash for Jaccard similarity [3], p-stable distribution LSH for ℓ_p distance [9], and more (Appendix F). In our work, we leverage LSH to construct similarity-preserving and space-efficient node representations for user stitching.

6 Conclusion

We have proposed a hash-based network representation learning framework for identity stitching called NODE2BITS. It is both time- and attribute-aware, while also deriving space-efficient sparse binary embeddings of nodes in large temporal heterogeneous networks. NODE2BITS uses the notion of feature-based temporal walks to capture the temporal and feature-based information in the data. Feature-based temporal walks are a generalization of walks that obey time while also incorporating features (as opposed to node IDs). Using these walks, NODE2BITS generates contexts/sequences of temporally valid feature values. Experiments on real-world networks demonstrate the utility of NODE2BITS as it outputs space-efficient embeddings that use orders of magnitude less space compared to the baseline methods while achieving better performance in user stitching. An

important practical consideration in the application of our work to user stitching is the balance of greater personalization with user privacy.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. IIS 1845491, Army Young Investigator Award No. W911NF1810397, an Adobe Digital Experience research faculty award, 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. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

1. Ahmed, N.K., Rossi, R.A., Zhou, R., Lee, J.B., Kong, X., Willke, T.L., Eldardiry, H.: Learning role-based graph embeddings. In: arXiv:1802.02896 (2018)
2. Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. TKDD **1**(1), 1–36 (2007)
3. Broder, A.Z., Glassman, S.C., Manasse, M.S., Zweig, G.: Syntactic clustering of the web. Computer Networks and ISDN Systems **29**(8), 1157–1166 (1997)
4. Cao, S., Lu, W., Xu, Q.: Deep neural networks for learning graph representations. In: AAAI. pp. 1145–1152 (2016)
5. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: STOC. pp. 380–388 (2002)
6. Christen, P.: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer (2012)
7. Cohen, W.W., Richman, J.: Learning to match and cluster large high-dimensional data sets for data integration. In: KDD. pp. 475–480 (2002)
8. Dasgupta, A., Gurevich, M., Zhang, L., Tseng, B., Thomas, A.O.: Overcoming browser cookie churn with clustering. In: WSDM. pp. 83–92 (2012)
9. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: SCG. pp. 253–262 (2004)
10. Dong, X.L., Naumann, F.: Data fusion: resolving data conflicts for integration. VLDB **2**(2), 1654–1655 (2009)
11. Dong, Y., Chawla, N.V., Swami, A.: metapath2vec: Scalable representation learning for heterogeneous networks. In: KDD. pp. 135–144 (2017)
12. Eckersley, P.: How unique is your web browser? In: International Symposium on Privacy Enhancing Technologies Symposium. pp. 1–18. Springer (2010)
13. Eirinaki, M., Vazirgiannis, M.: Web mining for web personalization. ACM Trans. Internet Technol. **3**(1), 1–27 (Feb 2003)
14. Getoor, L., Machanavajjhala, A.: Entity resolution for big data. In: KDD. pp. 1527–1527 (2013)
15. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: KDD. pp. 855–864 (2016)
16. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NIPS. pp. 1024–1034 (2017)

17. Heimann, M., Shen, H., Safavi, T., Koutra, D.: Regal: Representation learning-based graph alignment. In: CIKM. pp. 117–126 (2018)
18. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC. pp. 604–613 (1998)
19. Kim, S., Kini, N., Pujara, J., Koh, E., Getoor, L.: Probabilistic visitor stitching on cross-device web logs. In: WWW. pp. 1581–1589 (2017)
20. Kolb, L., Thor, A., Rahm, E.: Dedoop: Efficient deduplication with hadoop. VLDB **5**(12), 1878–1881 (Aug 2012)
21. Nguyen, G.H., Lee, J.B., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S.: Continuous-time dynamic network embeddings. In: WWW BigNet (2018)
22. Papadakis, G., Svirsky, J., Gal, A., Palpanas, T.: Comparative analysis of approximate blocking techniques for entity resolution. VLDB **9**(9), 684–695 (2016)
23. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: KDD (2014)
24. Ribeiro, L.F., Saverese, P.H., Figueiredo, D.R.: struc2vec: Learning node representations from structural identity. In: KDD. pp. 385–394 (2017)
25. Ricci, F., Rokach, L., Shapira, B.: Recommender Systems Handbook. Spr. (2011)
26. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: AAAI (2015), http://networkrepository.com
27. Rossi, R.A., Zhou, R., Ahmed, N.K.: Deep inductive network representation learning. In: WWW. pp. 953–960 (2018)
28. Saha Roy, R., Sinha, R., Chhaya, N., Saini, S.: Probabilistic deduplication of anonymous web traffic. In: WWW. pp. 103–104 (2015)
29. Shi, Y., Gui, H., Zhu, Q., Kaplan, L., Han, J.: Aspem: Embedding learning by aspects in heterogeneous information networks. In: SDM. pp. 144–152. SIAM (2018)
30. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: LINE: large-scale information network embedding. In: WWW (2015)
31. Tang, L., Liu, H.: Leveraging social media networks for classification. DMKD **23**(3), 447–478 (2011)
32. Wang, J., Shen, H.T., Song, J., Ji, J.: Hashing for similarity search: A survey. arXiv preprint arXiv:1408.2927 (2014)
33. Wang, Q., Wang, S., Gong, M., Wu, Y.: Feature hashing for network representation learning. In: IJCAI. pp. 2812–2818 (2018)