Succinct Graph Representations as Distance Oracles: An Experimental Evaluation

Arpit Merchant
University of Helsinki
arpit.merchant@helsinki.fi

Aristides Gionis
KTH Royal Institute of Technology
argioni@kth.se

Michael Mathioudakis
University of Helsinki
michael.mathioudakis@helsinki.fi

ABSTRACT

Distance oracles answer shortest-path queries between any pair of nodes in a graph. They are often built using succinct graph representations such as spanners, sketches, and compressors to minimize oracle size and query answering latency. Node embeddings, in particular, offer graph representations that place adjacent nodes nearby each other in a low-rank space. However, their use in the design of distance oracles has not been sufficiently studied.

In this paper, we empirically compare exact distance oracles constructed based on a variety of node embeddings and other succinct representations. We evaluate twelve such oracles along three measures of efficiency: construction time, memory requirements, and query-processing time over fourteen real datasets and four synthetic graphs. We show that distances between embedding vectors are excellent estimators of graph distances when graphs are well-structured, but less so for more unstructured graphs. Overall, our findings suggest that exact oracles based on embeddings can be constructed faster than multi-dimensional scaling (MDS) but slower than compressed adjacency indexes, require less memory than landmark oracles but more than sparsifiers or indexes, can answer queries faster than indexes but slower than MDS, and are exact more often with a smaller additive error than spanners (that have multiplicative error) while not being lossless like adjacency lists. Finally, while the exactness of such oracles is infeasible to maintain for huge graphs even under large amounts of resources, we empirically demonstrate that approximate oracles based on GOSH embeddings can efficiently scale to graphs of 100M+ nodes with only small additive errors in distance estimations.

PVLDB Reference Format:
Arpit Merchant, Aristides Gionis, and Michael Mathioudakis. Succinct Graph Representations as Distance Oracles: An Experimental Evaluation. PVLDB, 15(11): 2297 - 2306, 2022.
doi:10.14778/3551793.3551794

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://version.helsinki.fi/ads/distance_oracles.

1 INTRODUCTION

Distance oracles answer distance queries between any pair of nodes in a graph. To be usable in practice, they should process such queries efficiently, have low construction time, and require small amount of memory. In this paper, we evaluate the performance of various distance oracles and identify their relative strengths and weaknesses.

Distance queries are a fundamental primitive in applications such as generating unbiased samples for distance based experiments [61], distance-vector algorithms that must maintain precise routing tables [45], and identifying optimal pathways for efficient communications in metabolic neural receptors [11, 49]. Graphs in these domains have small-world characteristics and are bounded by a small diameter. Thus, relative errors can be significantly large thereby adversely affecting performance [11, 12]. Further, large graph size makes it computationally challenging to frequently compute exact distances at runtime with minimal latency using traditional shortest path algorithms such as breadth-first search.

To answer queries efficiently, distance oracles often make use of succinct graph representations [25, 26, 29]. These representations come in different forms, namely spanners [44], sketches [19], and compressors [22], while classic oracles rely on landmarks [54], and all-pairs shortest paths (APSP) algorithms leverage matrix-multiplication [20]. Some representations, such as compressors and APSP, are lossless, meaning that they return the exact distance for any source-target node pair. However, most types of spanners and oracles are lossy and only return approximate distance estimates.

Embeddings, i.e., mappings of nodes onto a low-rank vector space, offer an alternative approach. For instance, multi-dimensional scaling (MDS) preserves node distances while extracting a low-rank approximation of the distance matrix [33]. Moreover, node embeddings such as the spectral embedding minimize a global distortion measure [7]. Advances in deep learning have also led to a generation of random-walk embeddings that leverage skip-gram models to preserve local structure by capturing first- and second-order proximities [13], and graph neural networks (GNNs) that model higher-order neighborhoods [58, 62]. However, although node embeddings have been extensively developed recently, an evaluation of node embeddings as distance oracles is missing from the literature.

In this paper, we aim to fill this gap. To this end, we conduct an extensive comparative study of distance oracles based on a variety of succinct graph representations, with a focus on exact oracles. Each oracle is built offline and is employed to answer distance queries at runtime. An Oracle consists of two components: (i) a model and (ii) optionally, a set of exceptions. The model takes as input a pair of nodes and returns a (possibly approximate) estimate of their graph distance. To compute this estimate, the model makes use of a graph representation, such as a a node embedding or a spanner, which is constructed during the offline phase. For example, Oracles using node embeddings estimate the graph distance between two nodes from their distance in the embedding space, while Oracles based on other graph representations use specifically-tailored variants of

Proceedings of the VLDB Endowment, Vol. 15, No. 11 ISSN 2150-8097.
doi:10.14778/3551793.3551794

breadth-first-search (BFS). The set of exceptions stores the distances between those pairs of nodes that the model is unable to answer exactly. In practice, this takes the form of a lookup table with pairs of nodes as keys. At query time, given an input pair of nodes, the Oracle checks whether an entry for the given pair exists in its exceptions set. If so, it returns the corresponding distance value, otherwise it queries the model. Therefore, by construction, each Oracle is ensured to be exact under the above scheme, as long as the set of exceptions can be maintained. However, for very large graphs (i.e., graphs of tens of millions of nodes or more), computing such a set of exceptions in not an option, as enumerating all node pairs becomes infeasible even with large amounts of resources. In such cases, oracles are necessarily approximate.

Our experiments demonstrate that different Oracles offer different trade-offs between construction time, memory requirements (size of model and exceptions set), and query-processing time and no single Oracle is optimal across all measures. We evaluate Oracles on four synthetic graph families and fourteen real-world graph datasets. Our main findings are:

- Node embeddings offer excellent estimates of graph distances when graphs are highly structured, e.g., strongly regular or dense.
- Node embeddings are more efficient to construct than low-rank factorizations of the distance matrix such as MDS [33, 34]. However, MDS captures graph distances better.
- Node embeddings are significantly faster at processing distance queries than graph compression using DINT [47]. But DINT is lossless and extremely time-efficient to construct.
- Node embeddings are more accurate at estimating distances than TZ-oracles [54] and spanners [44], but are slower to construct.
- Only approximate Oracles based on Gosh [3] embeddings scale to graphs of 100M nodes within our resource budget.

2 BACKGROUND

Traditional approaches. Thorup and Zwick [54] construct a data structure of size $O(tn^{1+1/t})$, in preprocessing time $O(tmn^{1/t})$, to answer distance queries in time $O(t)$, with a distortion of up to $2t-1$, for any integer $t \geq 1$. Follow-up work offers variants that offer different trade-offs [1, 6, 16, 59]. We suggest Sommer [53] for a survey on such data structures.

Graph reductions. Spanners decrease the size of a graph while maintaining distance. A t-spanner of a graph G, for $t \geq 1$, is a subgraph H of G of equal size but fewer edges such that all pairwise distances are distorted by a factor of at most t [44]. They are closely associated with distance oracles and share the same existential size-distortion tradeoff via the Erdös girth conjecture [24]. Althöfer et al. [4] construct greedy $(2t-1)$-spanners of optimal size, while Chechik et al. [16] introduce fault-tolerant spanners that allow upto f edges failures at an additional f distortion factor. Please see Bodwin et al. [9] for a recent survey on spanners.

Another class of reductions focuses on graph compression using two heuristics, namely, homophily (similar nodes have similar neighbors) and locality (edges between node pairs are likely to point to other node pairs nearby). Compression involves a node reordering scheme and an encoding scheme where the choice of the former directly impacts the compression ratio achieved by the latter. Node reordering is proven to be **NP**-hard, but heuristics such as spectral [30] and recursive bisection [22] are known to perform well in practice. Cheng et al. [17] proposed compression schemes that explicitly target efficient processing of k-hop reachability queries.

Low-rank embeddings. Low-rank embeddings can be divided in two broad classes. First, are embeddings designed to reduce data dimensionality. This is achieved using projection methods, such as, Isomap [5], multi-dimensional scaling (MDS) [33], or Johnson-Lindenstrauss [32]. Alternatively, landmark embeddings store the distances from a small subset of landmark nodes to all other nodes in a graph. The choice of landmarks is determined via a reconstruction loss and distances are estimated using triangulation [31], packet routing [8], distance labels [25], or a trained neural network [50].

The second class comprises of modern node-embedding algorithms that represent nodes as low-dimensional vectors such that topological structures are preserved. Unsupervised embeddings such as NetSMF [48], or GraRep [14] capture first-order (e.g., adjacency) or higher-order (similarity between r-hop neighborhoods of nodes) proximity via matrix factorization or random-walk paths. Supervised embeddings, designed for attributed graphs from graph neural networks (GNNs), fuse node-level features with adjacency and ground-truth labels tasks such as node classification. We refer the reader to surveys on unsupervised embeddings [13], GNNs [58], representation learning [27], and benchmarking for node classification and link prediction [23, 28] for a comprehensive overview.

2.1 Choice of Representative Methods

In this paper, we place these disparate succinct graph representations into a common framework for distance oracles. Given their large number, we carefully choose representative methods to cover diverse approaches for our empirical study (Sections 5-6). We cover classical approaches with TZ Oracle [54], t-Spanner [44], and Landmarks [50]. To cover compression-based approaches that preserve graph distances, we include DisOracle [41] and DINT [47]. We include Spectral Embedding [43] and Multidimensional Scaling (MDS) [34] to cover traditional embedding methods that aim to preserve adjacency and distances, respectively. Moreover, we include NetSMF [48] and Asym-DNN [2] as modern node embedding methods that use random-walks and neural-networks. NetSMF generalizes previous approaches such as DeepWalk [46], which are thus not included in our experiments (e.g., NetSMF scales to larger graphs and produces embeddings of better quality than DeepWalk, as per Figure 2 and Table 5 of [48]). Lastly, we include FREDE [56] and GOSH [3] as efficient, state-of-the-art methods that produce node embeddings by factorizing node-similarity matrices. Since both FREDE and GOSH extend VERSE [55] and improve upon its performance, we do not include VERSE in our experiments (FREDE offers anytime computation and orders of speedup as per Figure 5 [56]; moreover, GOSH exhibits speedups in the range of 8.27-768.45x over VERSE on medium-size graphs for embeddings of similar quality, as per Table 6 [3], and unlike VERSE, GOSH better scales to large graphs of tens-of-million nodes as per Table 7 [3]).

3 SETUP

Let $G = (V, E)$ be a graph with $n = |V|$ nodes and $m = |E|$ edges. In this paper, we consider G to be undirected because we focus on methods designed for undirected graphs. Let $\mathbf{A}$ denote its adjacency

matrix with binary entries indicating the presence or absence of an edge. A path in a graph G is a finite sequence of nodes $v_1 \rightarrow \ldots \rightarrow v_{\ell+1}$ in which any two consecutive nodes are adjacent. The length of a path is the number of edges that appear in it, and the *graph distance* of two nodes is the minimum path length between them.

We consider the task of computing the graph distance for any pair of nodes $i, j \in V$. To address it, we consider a two-phase approach: an offline phase in which we construct a distance Oracle and an online phase wherein we use the Oracle to process distance queries. The two phases are described below.

Offline phase: The distance Oracle consists of two components namely, a Model and an (optional) Exceptions set. A Model M is equipped with (i) a succinct representation of the original graph G such as a spanner or a node embedding and (ii) a query-processing algorithm for estimating graph distances. In Section 4, we describe the different models that we study in this paper along with their query-processing algorithms. It should be noted, however, that some models are lossy and are formally known to have a multiplicative distortion factor denoted by t (e.g., TZ-Oracles [54] and t-spanners [44]). Therefore, they do not return the correct graph distance for some node pairs. For such node pairs, the exact distance may optionally be stored in a lookup table named Exceptions. If Exceptions are indeed maintained, it is guaranteed that the *exact* distance is available for all node pairs, even ones for which the Model returns an inexact answer. However, computing the Exceptions requires a full enumeration of node pairs. This becomes infeasible for very large graphs (node size 10M+) due to the quadratic (in graph size) number of node pairs.

Online phase: During this phase, the Oracle answers distance queries. If Exceptions are maintained, it first searches for the input node pair in the Exceptions set: if present, the Oracle returns the corresponding distance value from the lookup table; otherwise the query-processing algorithm of the Model is executed. Because the Exceptions contain the exact distance for all pairs of nodes that the Model does not answer correctly, the use of Exceptions ensures, by design, that Oracle always returns the correct answer. By contrast, if Exceptions are not maintained (e.g., in very large graph settings, where they are infeasible to compute), then query answering is uses only the Model's predictions, and is generally approximate.

In evaluating Oracles, we consider three key measures: (1) construction time, given adjacency matrix as input, (2) memory size, which is a sum of Model and Exceptions sizes, and (3) query processing time. For an Oracle to be competitive, it should demonstrate an advantage in at least one of these measures. In this paper, for the larger part, we focus on *exact* distance oracles, as we aim understand the trade-offs offered by different Oracles under the common requirement for exactness and a common budget of resources.

4 MODELS

In this section, we describe various Model choices and their query-processing algorithms, summarized in Table 1.

4.1 Traditional Models

Two natural baselines stand at two opposite extremes. The first is to simply store the graph in the adjacency-list format. This requires no preprocessing and produces a Model requiring size $O(m)$; any distance query can then be answered correctly using $O(m+n)$ operations (number of steps) using BFS. The second, is to use an All-Pairs Shortest Path (APSP) algorithm [15, 20] to preprocess the graph in $\tilde{O}(mn)$ time and store the $O(n^2)$ matrix holding the distances as the Model. Query processing takes $O(1)$ operations (lookup). While these baselines require no Exceptions to be separately stored, adjacency lists are slow to process queries at runtime and distance matrices are too large to compute and store efficiently. Thus, we consider the following traditional approximate Models:

- **TZ-Oracle** [54] creates a tree cover of the graph such that each node is contained in a small number of trees and returns the graph distance with a multiplicative distortion of at most t for any node pair. Query processing time depends on the operations required to identify the relevant tree and shortest path.
- **t-spanner** [44] returns a sparsified graph H using a randomized algorithm such that the distance between any node pair in H is at most t times the distance between the pair in G. The number of query operations is counted similarly to adjacency-list.
- **Landmarks** [50] chooses a small number of nodes l as landmarks and computes the graph distance from all nodes to each landmark. It maps distances in the embedding space to true graph distances using a feedforward neural network that requires a constant number of query operations independent of graph size.

4.2 Models based on node embeddings

We aim to evaluate how models based on node embeddings compare against models obtained from other succinct graph representations.

Definition 1 (Node Embeddings). *A node embedding is a mapping $\boldsymbol{U} : V \rightarrow \mathbb{R}^{n \times k}$ that maps each node i of a graph G to a k-dimensional vector $\boldsymbol{u}_i \in \mathbb{R}^k$ where $k \ll |V|$.*

Given embedding vectors $\mathbf{u}_i$ and $\mathbf{u}_j$ for nodes i and j, we denote by $\|\mathbf{u}_i - \mathbf{u}_j\|_p$ the ℓ_p-distance between i and j. We propose two models for a given node embedding: (i) GraphDT and (ii) NodeDT. Each of them takes the embedding distance $\|\mathbf{u}_i - \mathbf{u}_j\|_p$ as a feature and outputs an estimate $\hat{d}_G(i,j)$ of the graph distance $d_G(i,j)$.

GraphDT learns a single decision tree model for the entire graph, that is, for all $O(n^2)$ distinct node pairs. GraphDT captures global correlations between embedding distances and rescales them to appropriate graph distances. In contrast, NodeDT learns one decision tree model for each node $i \in V$. Thus, it has n decision trees but, depending on graph structure, the size of each node-specific tree can be small. The query processing time required to answer a query is defined as the number of decision tree operations (identifying a node-specific tree in NodeDT takes constant time).

Below, we briefly describe the embeddings. We write *GraphDT +X* or *NodeDT+X* to refer to the model that employs GraphDT or NodeDT, respectively, on top of embedding X.

- **Spectral Embedding** [43] is a matrix $\mathbf{U} \in \mathbb{R}^{n \times k}$ constructed by stacking eigenvectors corresponding to the k smallest eigenvalues of the unnormalized graph Laplacian, defined as $\mathbf{L} = D - \mathbf{A}$ where D is a diagonal matrix and $D(i,i)$ is the degree of node v_i. For very large graphs (10M+ nodes), we use GOSH's [3] parallelized approach to obtain approximate spectral embeddings.
- **NetSMF** [48] presents a scalable version of DeepWalk [46] by spectrally sparsifying a dense transition matrix $\mathbf{P}$ so as to enable

Table 1: Qualitative comparison of memory M, construction time T_C, and query time T_Q complexity of Models constructed from various succinct graph representations. t, d, k, δ, and c denote the distortion factor, average node degree in the graph, embedding dimension (where applicable), maximum label size, and number of CPU cores, respectively. Note, $k \ll n$.

Model	Lossless	Complexity		
		M	T_C	T_Q
Adjacency	✓	$O(m)$	$O(1)$	$O(n+m)$
APSP [20]	✓	$O(n^2)$	$\tilde{O}(mn)$	$O(1)$
TZ Oracle [54]	✗	$O(tn^{1+1/t})$	$O(tmn^{1/t})$	$O(t)$
t-Spanner [44]	✗	$\tilde{O}(t^c n^{1+1/t})$	$O(tmn^{1/t})$	$O(tn^{1+1/t})$
DINT [47]	✓	$O(n)$	$O(m)$	$O(n+m)$
DisOracle [41]	✓	$O(n^2)$	$O(\delta^2/c * m)$	$O(n)$
Landmark [50]	✗	$O(kn)$	$\tilde{O}(m+kn)$	$O(k^2 \log n)$
MDS [34]	✗	$O(kn^{1+1/k})$	$O(kmn^{1/k})$	$O(k)$
Spectral [43]	✗	$\tilde{O}(kn)$	$\tilde{O}(km)$	$O(k)$
NetSMF [48]	✗	$\tilde{O}(n^2)$	$\tilde{O}(n^2)$	$O(k)$
FREDE [56]	✗	$\tilde{O}(dn)$	$\tilde{O}(dn^2)$	$O(k)$
Asym-DNN [2]	✗	$\tilde{O}(kn)$	$\tilde{O}(kn^2)$	$O(k)$

fast sparse matrix factorization whilst maintaining the representative power of the learned embeddings from DeepWalk. Here, $\mathbf{P} = \log(\text{vol}(G)(\frac{1}{T}\sum_{r=1}^{T}(D^{-1}\mathbf{A})^r)D^{-1}) - \log b$ and $\text{vol}(G)$ is the volume of the graph, T represents the length of the random walk, and b is the number of negative samples.

- **FREDE** [56] individually processes the rows of a non-linearly transformed Personalized PageRank-based similarity matrix to sketch a close approximation of its optimal SVD. The resulting embedding has a closed-form solution, requires subquadratic (in n) space, and can be computed at any time, meaning that as more nodes are processed, its quality improves.
- **Asym-DNN** [2] models an edge as a function of node embeddings and information from sampled random walks with non-existent edges. It minimizes the likelihood of having observed the training graph $G = (V, E_{\text{train}})$ captured by the objective function $\prod_{i,j \in V} \sigma(g(i,j))^{R(i,j)}(1-\sigma(g(i,j)))^{\mathbb{I}[(i,j) \notin E_{\text{train}}]}$ where $\sigma(x)$ is the logistic function, $R(i,j)$ is the frequency with which i, j appear in simulated random walks, $\mathbb{I}$ is the indicator function, and g is a low-rank affine projection in the manifold space.

4.3 Multi-dimensional scaling

Multi-dimensional scaling (MDS) is a popular family of algorithms designed for projecting the $n \times n$ distance matrix into a low dimensional Euclidean space; given pairwise distances, reconstruct a map that preserves those distances. That is, MDS seeks to find points $\mathbf{u}_1, \ldots, \mathbf{u}_n \in \mathbb{R}^k$, for all $i \in [n]$, such that $d_G(i,j) \approx \|\mathbf{u}_i - \mathbf{u}_j\|_p$, for all node pairs (i,j). Given distance matrix $\mathbf{D}$, classical MDS constructs the Gram matrix $\mathbf{D}^\top\mathbf{D}$ and then applies double centering to get $\mathbf{B} = -\frac{1}{2}\mathbf{C}\mathbf{D}^\top\mathbf{D}\mathbf{C}$. Here, $\mathbf{C} = \mathbf{I} - \frac{1}{2}\mathbf{1}$, $\mathbf{I}$ is the identity matrix and $\mathbf{1}$ is the all-ones matrix. Then, MDS is computed as $n \times k$-dimensional embedding $\mathbf{U} = \mathbf{V}_{\{k\}}^\top \mathbf{S}_{\{k\}}$ where $\mathbf{S}_{\{k\}}$ represents a $k \times k$ diagonal matrix whose entries are the k largest in magnitude eigenvalues of $\mathbf{B}$ and $\mathbf{V}_{\{k\}}^\top$ are the corresponding eigenvectors. At query time, the graph distance between a node pair is estimated by the ℓ_p norm between the corresponding row vectors of $\mathbf{U}$ and is thus only dependent on k. We use (i) sketching to reduce dimensionality, (ii) block-striped cyclic decomposition for parallelizing matrix multiplication [57], and (ii) a parallelizable inverse iteration algorithm for approximately computing a partial eigendecomposition [21].

4.4 Compressed Adjacency Indexes

Compressed adjacency indexes minimize the number of bits used to store the topological information of the graph. We consider two schemes namely, (i) DINT [47], an inverted index, and (ii) DisOracle [41] which relies on topology based distance labeling. The core idea behind these approaches is the same. The graph is processed to define a node ordering, i.e. assigning labels or identifiers to nodes such as placing topologically similar nodes nearby in the resulting order, in a way that optimizes compression. Choosing the right node ordering can result in a significantly higher compression ratio [22]. Then, an encoding scheme is designed to build a data structure (e.g. dictionary) that can minimize the number of bits needed to encode adjacency information among sets of nodes, based on the node order and a ranking function. Last, a query answering algorithm traverses the index to compute the source-target distance.

- **DINT** [47] We use spectral ordering wherein nodes are arranged in increasing value in the second smallest Laplacian eigenvector, after experimentation with different node orderings. Spectral ordering places adjacent nodes within close proximity in the index, a property that leads to consistently good performance in comparison with baseline node orderings (e.g., random). Next, the index is constructed using a single-packed rectangular dictionary of node IDs as integer sequences. This enables fixed-to-fixed decoding which executes a copying operation of constant predetermined length from the index to the output buffer and is thus extremely fast. Using DINT, graph distance between a pair of nodes is computed via BFS traversal (note: BFS for DINT is implemented similarly to BFS for adjacency list, with appropriate decoding for the nodes reached at each BFS expansion step).
- **DisOracle** [41] uses 2-hop labeling to create hub nodes, similar to landmarks, and assigns the distance to the hub nodes as labels to each other node in the graph. 2-hop labeling ensures that the shared hubs of each pair of nodes have at least one common node. Finding the optimal labeling is NP-hard and the resulting index can be large (quadratic in the number of nodes). DisOracle transforms the node order dependence from adjacency to distance information in a parallelized label propagation based manner and uses equivalence relation elimination to prune redundant labels (recognized as PSL+). This speeds up construction and reduces index size compared to landmark labeling and 2-hop approaches.

We note that DINT is originally presented as a compressed adjacency index in an *information retrieval* setting, used to retrieve those documents that contain a specified term – but the data structure can equally well be used as a compressed adjacency index in our setting, to retrieve nodes connected to a specified node. DisOracle offers a parallelized variant of a compressed index compared to

the sequential nature of DINT. They require a larger index but offer a faster query processing time. Tree decomposition based methods are another variant that specialize in identifying the core-fringe structure of graphs and then create an index on these two separate parts. Since we consider the general graph case, we do not directly compare against them. We refer the reader to a recent experimental study on distance labeling algorithms for the same [42].

5 EXPERIMENTS ON SYNTHETIC GRAPHS

In this section, we demonstrate the impact of graph structure on the performance of embedding distances as estimates for graph distances. We use GraphDT+Spectral as Oracle for four synthetic graph families, of varying regularity in their structure: (i) connected caveman (CC), (ii) Barabási-Albert (BA), (iii) Watts-Strogatz (WS), and (iv) Erdös-Rényi (ER). For each family, we construct a toy graph instance with $n = 200$ nodes and its spectral embedding with dimension $k = 2$. We also compute the graph distance and the embedding distance (ℓ_2-norm between embedding vectors), for all $n(n-1)/2$ distinct node pairs, and train GraphDT. Further implementation details and results are available in the supplementary material.

Figure 1 reports the results on synthetic data. The first row of plots displays the distribution of embedding distance as a function of graph distance for the four synthetic graph families. The second row shows the number of decision tree operations performed by GraphDT as a function of graph distance. We find that embedding distances are a good proxy for graph distances. GraphDT is able to recover exact graph distances for 99.47, 99.46, 99.39, and 95.63 percent of node pairs for CC, BA, WS, and ER graphs, respectively. From the first row of plots, we observe that the overlap between boxes increases as we go from CC to ER graphs. This implies that the gap in the embedding distance between nodes at different graph distances reduces as the graph becomes less regular and less dense. Nodes at distance 2 and 4 are well-separated in the embedding space for CC and BA compared to WS and ER, thus leading to more errors. From the plots in the second row, we observe the increasing number of decision tree operations required for estimation. CC requires 5 operations on average while ER requires 32. Crucially, GraphDT displays a small average additive error of 1 (e.g., it estimated the path length to be either 1, 2, or 3 when the actual graph distance was 2) in all four graphs. Thus, we conclude that embedding-based Oracles can be very effective for well-structured graphs.

6 EXPERIMENTS ON REAL GRAPHS

6.1 Datasets and Setup

In this section, we study the advantages and disadvantages of using node embeddings versus other succinct representations in constructing exact Oracles. Table 2 provides an overview of dataset statistics. A longer description is offered in the supplementary material. We explicitly model all graphs as undirected including wiki-Vote, web-BerkStan, and Twitter which are originally directed.

Resource Budget. All experiments that follow are performed under the same budget of computational resources. Parallelized code implementations make use of Intel(R) Xeon(R) CPU E7-8890 v4 @ 2.20GHz with 128 cores and Tesla P100-PCIE GPU with 128 cores (where applicable). For each Oracle, we allow 24 hours each for construction and exceptions, and up to 500GB RAM. Following Section 3, for each graph and for a given Model, we first attempt to build an *exact* Oracle (i.e. an appropriate set of Exceptions alongside the Model to guarantee exactness). If the resources do not suffice to guarantee exactness (i.e., to compute and maintain an Exceptions), then instead we maintain an *approximate* Oracle– i.e., the Oracle produces approximate distance estimates using only the given Model and without making use of Exceptions.

Table 2: Dataset statistics: number of nodes $|V|$, number of edges $|E|$, average degree d_{avg}, average clustering coefficient C, and density ρ (number of edges as fraction of node pairs) of the graph, respectively. Density is defined as $|E|/(|V|(|V|-1))$. † denotes datasets with directed edges originally. However, we explicitly model all datasets as undirected.

Graph	Size		Properties		
	$\|V\|$	$\|E\|$	d_{avg}	C	$\rho(\times 10^{-5})$
cora [52]	2.5K	5.1K	4.1	0.24	200
twitch-RU [40]	4.4K	37K	17.0	0.17	400
twitch-FR [40]	6.5K	110K	34.4	0.22	500
wiki-Vote† [37]	7.1K	100K	28.5	0.14	400
twitch-DE [40]	9.5K	150K	32.3	0.20	300
ca-CondMat [38]	21K	91K	8.6	0.64	40
email-Enron [39]	34K	180K	10.7	0.51	70
blogcatalog [51]	89K	2.1M	47.2	0.35	53
loc-gowalla [18]	200K	950K	9.7	0.24	48
com-DBLP [60]	320K	1M	6.6	0.63	20
web-BerkStan† [39]	650K	6.6M	20.1	0.61	3.1
roadNet-PA [39]	1.1M	1.5M	2.8	0.05	4.0
Twitter† [35]	41.6M	1.4B	70.51	–	0.084
UK Domain [10]	105M	3.3B	62.8	0.03	0.029

We find that these resource-budget constraints induce a separation of the datasets into two groups. For the first group, the resource budget suffices to build an *exact* Oracle for each Model of Table 1. The first group consists of roadNet-PA (1.1M nodes) and all smaller graphs. The second group consists of graphs Twitter (41M nodes) and UK Domain (105M nodes). For the second group, the resource budget does not suffice to build an exact Oracle. We refer to the first group as the "small-to-large" and the second group as the "very large" graphs and split the presentation of results accordingly: Section 6.2 presents results on *exact* Oracles for the smaller graphs and Section 6.3 approximate Oracles on the very large ones.

Implementation Details. Our experimentation has three primary computational bottlenecks. (i) Finding graph distances between distinct node pairs. This takes a few seconds for the smallest graph (cora), upto 27 hours for the 1.1M-node graph (roadNet-PA), and 23 hours for 100K nodes for very large graphs. (ii) Constructing succinct representations. We augment publicly-available implementations of TZ-Oracle, t-spanner, NetSMF, FREDE, and Asym-DNN for embedding distances, while DINT is augmented with Algorithm 1 for estimating distances. (iii) Fitting decision tree(s) using $O(n^2)$ training samples. NodeDT is constructed by parallelizing over individual nodes while GraphDT is parallelized over multiple CPU

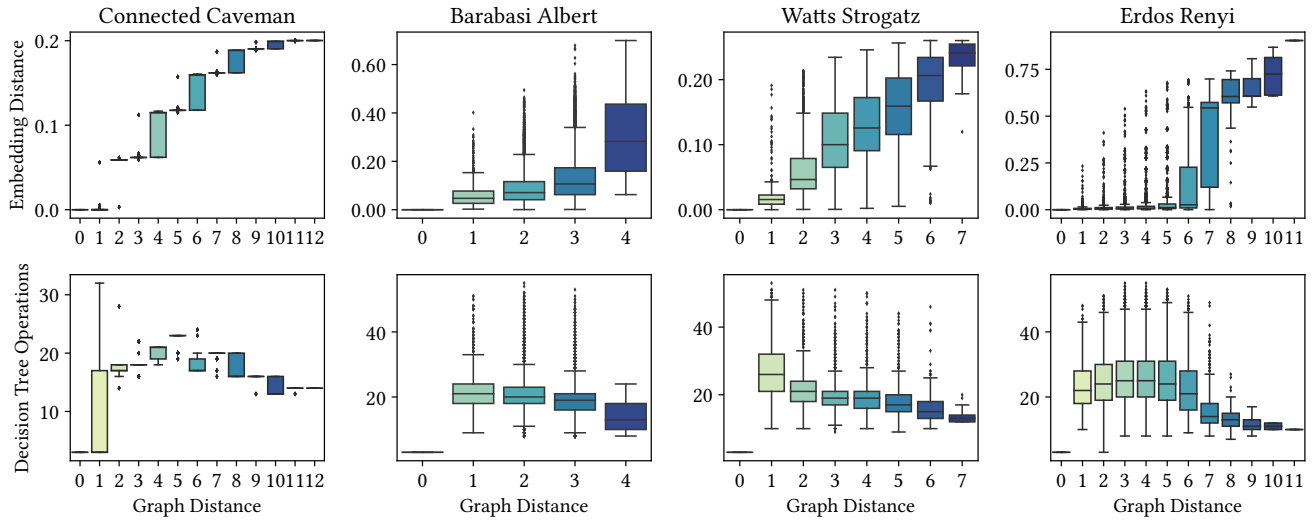

Figure 1: Results of GraphDT on four synthetic graphs. The plots in the first and second row display the distance in the embedding space and the number of decision tree operations (required for distance estimation) as a function of the actual graph distance, respectively.

cores with a balanced workload determined by recursively splitting a root histogram of the data into child histograms. In the case of MDS, for smaller graphs such as Cora whose Gram Matrix **B** can be held in memory, we use the Lanczos algorithm [36] for approximately computing the k largest eigenvectors. For larger graphs, we use SCALAPACK's parallelized implementations for matrix multiplication and approximate eigendecomposition (PxSYTRD).

Parameter Settings. We construct t-spanners with $t = 10$. We set the embedding dimension to $k = 128$ for all algorithms because larger k returned marginal improvements at the cost of increased construction time. We construct DINT using a single packed dictionary encoding. We seek hyperparameters that lead to good performance using manual grid search. Lastly, we choose a set of 100K random node pairs as queries for evaluating query processing time.

6.2 Exact Oracles on small-to-large Graphs

Figure 2 depicts the relative benefits and limitations of Oracles with respect to memory requirements and query processing time, using the adjacency list as a baseline. For each dataset, we compute the memory required to store the adjacency list (in MB) and the query processing time (in number of operations) it needs to answer 100K queries. Each marker in Figure 2 reports these two quantities for other Oracles as a fraction of the respective quantities for the adjacency list. Table 3 displays the size of Exceptions for lossy Oracles. We describe our key observations below.

Node-Embedding Oracles process distance queries faster than DINT, but require more memory and are slower to construct. For instance, we observe that DINT requires 0.65×operations compared to Adjacency-List (*email-Enron*), while GraphDT needs 0.42× since, in effect, DINT answers queries via BFS. This difference is more pronounced (0.87× vs. 0.11×) for *roadNet-PA* because of its grid-like structure, large size, and low average degree. However, DINT is designed to be extremely memory-efficient and scales very well with graph size. The C++ implementation of DINT takes between 1.2 (*cora*) and 62.8 (*web-BerkStan*) seconds to create the index, with size between 0.001 MB and 16.6 MB and no additional space for exceptions since its compression is lossless.

In contrast, the combined time to construct node embeddings and train decision trees for NodeDT drastically increases with graph size. Embedding a dense graph such as *com-DBLP* with NetSMF takes 7.3 hours. Also, large imbalance in actual graph distances leads to instability and fluctuation in the depth and size of learned decision trees. Even after enforcing shallow depth and high compression, the memory required for NodeDT (n decision trees along with Exceptions) is impractically large compared to Adjacency-List; e.g., 125×for small graphs like *twitch-FR* and 7646×for larger graphs like *web-BerkStan*). As a consequence of the shallow depth, we find that the number of Exceptions required by NodeDT+FREDE slightly exceeds that of GraphDT+FREDE in the case of *loc-gowalla*. This may be rectified by allowing greater depth. But we see the benefits of NodeDT at query time when it takes 0.15 and 0.026 fraction of operations needed by Adjacency-List (for *email-Enron* and *roadNet-PA*, resp.) because the size of each node-specific decision tree is small and accessing it takes constant time.

Node-Embedding Oracles require more memory and processing time than MDS, but are faster to construct. For instance, GraphDT+NetSMF takes less than half the time taken by MDS on medium-sized graphs like *loc-gowalla* whereas incremental approaches like FREDE can be upto 19× faster. For larger graphs such as *web-BerkStan*, MDS becomes prohibitively expensive since it requires multiplication and factorization of two large dense matrices. However, query answering for MDS is independent of n and m and is thus extremely fast (constant time) requiring 0.035× (*loc-gowalla*) or 0.044× (*web-BerkStan*) number of operations compared

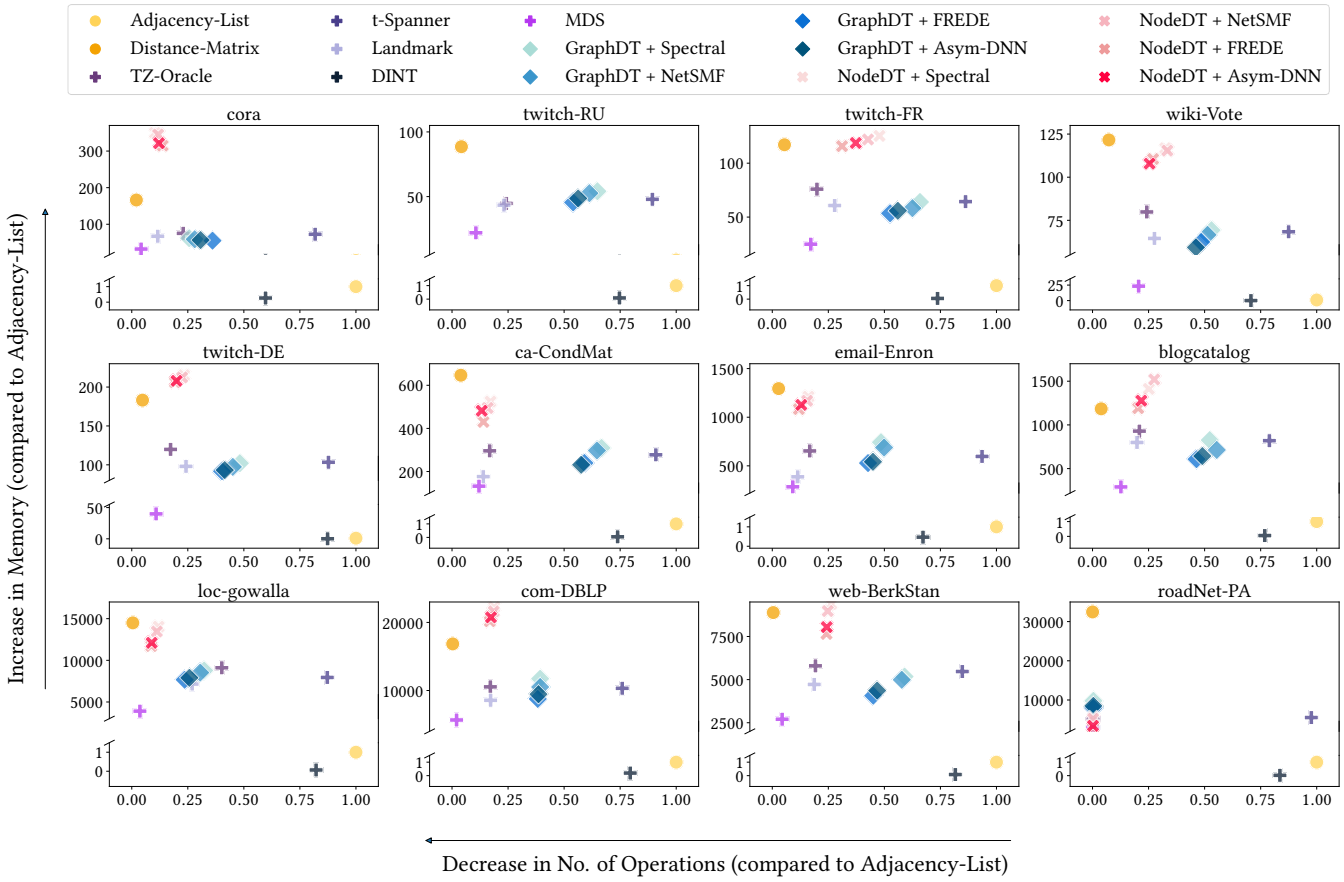

Figure 2: Query processing time (number of operations) and memory requirements (MBs needed for storing Model and Exceptions) for different Oracles, reported as multiplicative factors over the respective quantities of Adjacency List.

to Adjacency-List while GraphDT+FREDE takes 0.24× and 0.46× operations, respectively. We find that the size of MDS's Exceptions set is <20% of total node pairs for smaller graphs like *cora* or *twitch-FR* (cf. Table 3) and <30% for larger graphs like *com-DBLP*. This is less than that of other node embeddings since they do not directly preserve distances. We note that incorrect estimates by embedding-Oracles are off by only a small additive factor for all datasets.

Node-Embedding Oracles use fewer exceptions than t-Spanner [44] and TZ-Oracle [54]. To observe a non-trivial sparsification of the graph, distortion (t) needs to be set as high as 10. As a result, the corresponding size of their Exceptions set can be as high as 75% of distinct node pairs (cf. *blogcatalog* in Table 3). This distortion factor is multiplicative leading to larger errors in the estimates compared to GraphDT or NodeDT which have small additive error. Yet, this does not significantly improve query answering time for t-spanner either, which can be nearly four times as expensive as NodeDT for graphs with higher density and large average degree such as *twitch-DE* (requiring 0.87X and 0.22X operations required by Adjacency-List, resp.). Their advantage lies in the faster construction time compared to embedding-based Oracles.

We emphasize that, upto this point, our experiments do not identify a single Oracle that dominates others for all measures of interests. Our results provide a guide for users to determine the best Oracle given their construction, memory, and processing budgets.

6.3 Approximate Oracles on very large Graphs

In extending the evaluation of exact Oracles to very large graphs, namely Twitter and UK Domain, we came across two major bottlenecks: first, all-pairs distance computation proved intractable, with running times that exceeded our resource budget (24 hours), by orders of magnitude in our estimate; second, most evaluated models, including NetSMF and Asym-DNN, did not scale to very large graphs, with their computation exceeding our resource budget (24 hours). As FREDE provides any-time computation, we were able to train it for a very small number of epochs, which were not sufficient to produce stabilized embeddings and good distance estimates.

In light of these bottlenecks, we extend our evaluation to an approximate setting for very large graphs with GOSH [3] embeddings, which was the one model that scaled well enough to provide reasonably good results. We use a random subset of node pairs that is large enough to allow computation within our resource budget, and train on them a decision tree model. The training and testing

Table 3: Size of the Exceptions set for all lossy Oracles as a fraction of total distinct node pairs. Numbers in blue represent the best lossy Oracle and underlined values represent second-best within margin of error. Adjacency-List, Distance-Matrix, and DINT are lossless and require no exceptions to be stored.

Dataset	TZ-Oracle	t-spanner	Landmark	MDS	GraphDT				NodeDT			
					Spectral	NetSMF	FREDE	Asym-DNN	Spectral	NetSMF	FREDE	Asym-DNN
cora	0.495	0.481	0.433	0.174	0.415	0.408	0.368	0.377	0.352	0.325	0.271	0.279
twitch-RU	0.515	0.528	0.447	0.189	0.478	0.450	0.392	0.412	0.432	0.409	0.343	0.382
twitch-FR	0.551	0.533	0.474	0.196	0.492	0.480	0.414	0.423	0.498	0.482	0.297	0.316
wiki-Vote	0.585	0.561	0.521	0.188	0.516	0.520	0.491	0.480	0.466	0.487	0.358	0.340
twitch-DE	0.589	0.563	0.528	0.215	0.496	0.483	0.435	0.443	0.461	0.497	0.342	0.333
ca-CondMat	0.439	0.429	0.271	0.203	0.429	0.417	0.321	0.315	0.278	0.259	0.198	0.196
email-Enron	0.472	0.461	0.298	0.219	0.476	0.431	0.301	0.314	0.388	0.353	0.204	0.208
blogcatalog	0.754	0.692	0.674	0.245	0.592	0.532	0.467	0.496	0.453	0.541	0.390	0.397
loc-gowalla	0.597	0.548	0.493	0.269	0.457	0.416	0.390	0.402	0.462	0.423	0.394	0.409
com-DBLP	0.627	0.613	0.508	0.283	0.518	0.485	0.412	0.397	0.462	0.444	0.319	0.331
web-BerkStan	0.681	0.644	0.531	0.304	0.576	0.518	0.449	0.468	0.515	0.488	0.325	0.343
roadNet-PA	0.161	0.177	0.093	0.117	0.362	0.233	0.225	0.233	0.184	0.149	0.061	0.072

Table 4: Performance of GOSH Oracle: acuracy on train/test set (Acc.), average additive error in distance estimations (E_{avg}), memory (M in GB), construction time T_C, and average query processing time (T_Q in seconds), respectively.

Dataset	Acc.	E_{avg}	M	T_C	T_Q
roadNet-PA	73.56 / 70.45	1.53	1.3G	1.9H	2.3E-2
Twitter	84.67 / 83.71	1.26	21G	5.2H	4.8E-5
UK Domain	69.42 / 67.18	0.97	48G	8.3H	9.7E-6

set is created by randomly choosing a set of S source nodes and T_s destination nodes for each $s \in S$. Further experimental details are provided in the supplementary material. Moreover, we employ GOSH to obtain embeddings in $k = 128$ dimensions.

We find that given the strong power-law structure of the graph which is captured by the training data, the approximate GOSH-based Oracle learns the mapping from embedding distance to graph distance with good accuracy (84.71% and 69.51% on Twitter and UK Domain, respectively). The learned decision tree model itself requires 3.57MB and 7.13MB disk space, and 6.74 and 20.33 average number of operations for Twitter and UK Domain, respectively.

7 CONCLUSION

In this paper, we presented an extensive experimental evaluation of succinct graph representations as exact distance oracles. We comparatively analysed traditional approaches using spanners, TZ-oracles, distance matrices and modern approaches using compressed adjacency indexes and node embeddings on four synthetic and twelve real datasets. For embeddings, we defined two models, GraphDT that fits a single decision tree for the entire graph and NodeDT that fits one decision tree per node to learn a mapping from the embedding distance to the graph distance, respectively. We find that these models are excellent estimators of graph distances when graphs are well-structured, but their performance degrades when graph structure is more random. The choice of the embedding impacts the number of exceptions needed to be stored. NodeDT requires fewer exceptions to be stored and less query processing time compared to GraphDT, but significantly more memory, thus making it impractical for common use. Moreover, we find that while no single oracle is optimal across all efficiency measures, node embedding based Oracles are upto 19 times faster than MDS, require up to 2 times less memory than approximate distance-preserving data structures, up to 23 times less processing time than compressed indexes, and are exact up to 1.7 times more often than spanners. Crucially, approximate oracles based on scalable GOSH embeddings estimate graph distances with only small additive errors and can be efficiently constructed for large graphs (100M+ nodes).

7.1 Limitations and Future Work

There are two primary limitations of the distance oracles. (1) With the exception of MDS, node embedding algorithms do not explicitly optimize for preserving graph distances in their objective functions which results in reduced capacity to recover exact distances. (2) Computing all-pairs-distances and succinct representations such as node embeddings are inherently non-scalable tasks for very large graphs. This makes the construction of exact distance oracles intractable. We identify these as directions for future research.

ACKNOWLEDGMENTS

Arpit Merchant would like to thank Ananth Mahadevan and Sachith Pai for useful discussions on experiment design. Aristides Gionis is supported by the Academy of Finland projects AIDA (317085) and MLDB (325117), the ERC Advanced Grant REBOUND (834862), the EC H2020 RIA project SoBigData++ (871042), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. Michael Mathioudakis is supported by the University of Helsinki and the Academy of Finland Projects MLDB (322046) and HPC-HD (347747).

REFERENCES

[1] Ittai Abraham and Cyril Gavoille. 2011. On Approximate Distance Labels and Routing Schemes with Affine Stretch. In *Distributed Computing*, David Peleg (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 404–415.

[2] Sami Abu-El-Haija, Bryan Perozzi, and Rami Al-Rfou. 2017. Learning Edge Representations via Low-Rank Asymmetric Projections. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (Singapore, Singapore) *(CIKM '17)*. Association for Computing Machinery, New York, NY, USA, 1787–1796. https://doi.org/10.1145/3132847.3132959

[3] Taha Atahan Akyildiz, Amro Alabsi Aljundi, and Kamer Kaya. 2020. GOSH: Embedding Big Graphs on Small Hardware. In *49th International Conference on Parallel Processing - ICPP* (Edmonton, AB, Canada) *(ICPP '20)*. Association for Computing Machinery, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.1145/3404397.3404456 Nominated for the best-paper award.

[4] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. 1993. On sparse spanners of weighted graphs. *Discrete & Computational Geometry* 9, 1 (1993), 81–100.

[5] Mukund Balasubramanian, Eric L Schwartz, Joshua B Tenenbaum, Vin de Silva, and John C Langford. 2002. The isomap algorithm and topological stability. *Science* 295, 5552 (2002), 7–7.

[6] Surender Baswana, Akshay Gaur, Sandeep Sen, and Jayant Upadhyay. 2008. Distance Oracles for Unweighted Graphs: Breaking the Quadratic Barrier with Constant Additive Error. In *Automata, Languages and Programming*, Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 609–621.

[7] Mikhail Belkin and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation* 15, 6 (2003), 1373–1396.

[8] Yakir Berchenko and Mina Teicher. 2009. Graph Embedding through Random Walk for Shortest Paths Problems. In *Stochastic Algorithms: Foundations and Applications*, Osamu Watanabe and Thomas Zeugmann (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 127–140.

[9] Greg Bodwin, Michael Dinitz, and Caleb Robelle. 2021. Optimal Vertex Fault-Tolerant Spanners in Polynomial Time. In *Proceedings of the Thirty-Second Annual ACM-SIAM Symposium on Discrete Algorithms* (Virtual Event, Virginia) *(SODA '21)*. Society for Industrial and Applied Mathematics, USA, 2924–2938.

[10] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. 2004. UbiCrawler: A Scalable Fully Distributed Web Crawler. *Software: Practice & Experience* 34, 8 (2004), 711–726.

[11] Ed Bullmore and Olaf Sporns. 2009. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature reviews neuroscience* 10, 3 (2009), 186–198.

[12] Julia M Burkhart, Marc Vaudel, Stepan Gambaryan, Sonja Radau, Ulrich Walter, Lennart Martens, Jörg Geiger, Albert Sickmann, and René P Zahedi. 2012. The first comprehensive and quantitative analysis of human platelet protein composition allows the comparative analysis of structural and functional pathways. *Blood, The Journal of the American Society of Hematology* 120, 15 (2012), e73–e82.

[13] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[14] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. GraRep: Learning Graph Representations with Global Structural Information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (Melbourne, Australia) *(CIKM '15)*. Association for Computing Machinery, New York, NY, USA, 891–900. https://doi.org/10.1145/2806416.2806512

[15] Timothy M Chan. 2010. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.* 39, 5 (2010), 2075–2089.

[16] Shiri Chechik. 2014. Approximate Distance Oracles with Constant Query Time. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing* (New York, New York) *(STOC '14)*. Association for Computing Machinery, New York, NY, USA, 654–663. https://doi.org/10.1145/2591796.2591801

[17] James Cheng, Zechao Shang, Hong Cheng, Haixun Wang, and Jeffrey Xu Yu. 2014. Efficient processing of k-hop reachability queries. *The VLDB journal* 23, 2 (2014), 227–252.

[18] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. 2011. Friendship and Mobility: User Movement in Location-Based Social Networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Diego, California, USA) *(KDD '11)*. Association for Computing Machinery, New York, NY, USA, 1082–1090. https://doi.org/10.1145/2020408.2020579

[19] Edith Cohen and Haim Kaplan. 2007. Summarizing Data Using Bottom-k Sketches. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing* (Portland, Oregon, USA) *(PODC '07)*. Association for Computing Machinery, New York, NY, USA, 225–234. https://doi.org/10.1145/1281100.1281133

[20] Camil Demetrescu and Giuseppe F Italiano. 2004. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)* 51, 6 (2004), 968–992.

[21] Inderjit Singh Dhillon. 1997. *A new O (N (2)) algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem*. University of California, Berkeley, Berkeley, CA, USA.

[22] Laxman Dhulipala, Igor Kabiljo, Brian Karrer, Giuseppe Ottaviano, Sergey Pupyrev, and Alon Shalita. 2016. Compressing Graphs and Indexes with Recursive Graph Bisection. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1535–1544. https://doi.org/10.1145/2939672.2939862

[23] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2020. Benchmarking Graph Neural Networks. *CoRR* abs/2003.00982 (2020), 47. https://arxiv.org/abs/2003.00982

[24] Paul Erdös. 1965. On some extremal problems in graph theory. *Israel Journal of Mathematics* 3, 2 (1965), 113–116.

[25] Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. 2004. Distance labeling in graphs. *Journal of Algorithms* 53, 1 (2004), 85–112.

[26] James D. Guyton and Michael F. Schwartz. 1995. Locating Nearby Copies of Replicated Internet Servers. *SIGCOMM Comput. Commun. Rev.* 25, 4 (oct 1995), 288–298. https://doi.org/10.1145/217391.217463

[27] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. , 24 pages. https://doi.org/10.48550/ARXIV.1709.05584

[28] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[29] Bradley Huffaker, Marina Fomenkov, Daniel J Plummer, David Moore, Kimberly Claffy, et al. 2002. Distance metrics in the Internet. In *Proc. of IEEE international telecommunications symposium (ITS)*. sn, IEEE, New York, NY, USA, 20.

[30] Martin Juvan and Bojan Mohar. 1992. Optimal linear labelings and eigenvalues of graphs. *Discrete Applied Mathematics* 36, 2 (1992), 153–168.

[31] Jon Kleinberg, Aleksandrs Slivkins, and Tom Wexler. 2004. Triangulation and embedding using small sets of beacons. In *45th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, New York, NY, USA, 444–453.

[32] Felix Krahmer and Rachel Ward. 2011. New and improved Johnson–Lindenstrauss embeddings via the restricted isometry property. *SIAM Journal on Mathematical Analysis* 43, 3 (2011), 1269–1281.

[33] Joseph B Kruskal. 1978. *Multidimensional scaling*. Number 11 in 1. Sage, Washington, DC, USA. https://doi.org/10.4135/9781412985130

[34] Joseph B Kruskal and Judith B Seery. 1980. Designing network diagrams. In *Proceedings of the First General Conference on Social Graphics*. US Department of the Census, United States Government Printing Office, Washington, DC, USA, 22–50.

[35] Jérôme Kunegis. 2013. KONECT: The Koblenz Network Collection. In *Proceedings of the 22nd International Conference on World Wide Web* (Rio de Janeiro, Brazil) *(WWW '13 Companion)*. Association for Computing Machinery, New York, NY, USA, 1343–1350. https://doi.org/10.1145/2487788.2488173

[36] Richard B Lehoucq, Danny C Sorensen, and Chao Yang. 1998. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods*. SIAM, Philadelphia, PA, USA.

[37] Jure Leskovec, Daniel Huttenlocher, and Jon Kleinberg. 2010. Signed Networks in Social Media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 1361–1370. https://doi.org/10.1145/1753326.1753532

[38] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 2–es.

[39] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. 2009. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics* 6, 1 (2009), 29–123.

[40] Jundong Li, Xia Hu, Jiliang Tang, and Huan Liu. 2015. Unsupervised Streaming Feature Selection in Social Media. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management* (Melbourne, Australia) *(CIKM '15)*. Association for Computing Machinery, New York, NY, USA, 1041–1050. https://doi.org/10.1145/2806416.2806501

[41] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2019. Scaling Distance Labeling on Small-World Networks. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1060–1077. https://doi.org/10.1145/3299869.3319877

[42] Ye Li, Leong Hou U, Man Lung Yiu, and Ngai Meng Kou. 2017. An experimental study on hub labeling based shortest path algorithms. *Proceedings of the VLDB Endowment* 11, 4 (2017), 445–457.

[43] Bin Luo, Richard C Wilson, and Edwin R Hancock. 2003. Spectral embedding of graphs. *Pattern recognition* 36, 10 (2003), 2213–2230.

[44] David Peleg and Alejandro A Schäffer. 1989. Graph spanners. *Journal of graph theory* 13, 1 (1989), 99–116.

[45] Charles E Perkins and Elizabeth M Royer. 1999. Ad-hoc on-demand distance vector routing. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*. IEEE, IEEE, New York, NY, USA, 90–100.

[46] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, New York, USA) *(KDD '14)*. Association for Computing Machinery, New York, NY, USA, 701–710. https://doi.org/10.1145/2623330.2623732

[47] Giulio Ermanno Pibiri, Matthias Petri, and Alistair Moffat. 2019. Fast Dictionary-Based Compression for Inverted Indexes. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (Melbourne VIC, Australia) *(WSDM '19)*. Association for Computing Machinery, New York, NY, USA, 6–14. https://doi.org/10.1145/3289600.3290962

[48] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. 2019. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 1509–1520. https://doi.org/10.1145/3308558.3313446

[49] Syed Asad Rahman and Dietmar Schomburg. 2006. Observing local and global properties of metabolic pathways:âĂŸload pointsâĂŹ and âĂŸchoke pointsâĂŹ in the metabolic networks. *Bioinformatics* 22, 14 (2006), 1767–1774.

[50] Fatemeh Salehi Rizi, Joerg Schloetterer, and Michael Granitzer. 2018. Shortest path distance approximation using deep learning techniques. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, IEEE, New York, NY, USA, 1007–1014.

[51] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2019. Multi-scale Attributed Node Embedding. arXiv:1909.13021 [cs.LG]

[52] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[53] Christian Sommer. 2014. Shortest-path queries in static networks. *ACM Computing Surveys (CSUR)* 46, 4 (2014), 1–31.

[54] Mikkel Thorup and Uri Zwick. 2005. Approximate distance oracles. *Journal of the ACM (JACM)* 52, 1 (2005), 1–24.

[55] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. VERSE: Versatile Graph Embeddings from Similarity Measures. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 539–548. https://doi.org/10.1145/3178876.3186120

[56] Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller. 2021. FREDE: anytime graph embeddings. *Proceedings of the VLDB Endowment* 14, 6 (2021), 1102–1110.

[57] Robert A Van De Geijn and Jerrell Watts. 1997. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience* 9, 4 (1997), 255–274.

[58] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

[59] Christian Wulff-Nilsen. 2016. *Approximate Distance Oracles with Improved Query Time*. Springer New York, New York, NY, 94–97. https://doi.org/10.1007/978-1-4939-2864-4_568

[60] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42, 1 (2015), 181–213.

[61] Xiaohan Zhao, Alessandra Sala, Haitao Zheng, and Ben Y. Zhao. 2011. Efficient shortest paths on massive social graphs. In *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*. IEEE, New York, NY, USA, 77–86.

[62] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81.