On the Embeddability of Random Walk Distances

[†]Xiaohan Zhao, [†]Adelbert Chang, [‡]Atish Das Sarma, [†]Haitao Zheng, [†]Ben Y. Zhao
[†]Department of Computer Science, U. C. Santa Barbara [‡]eBay Inc.
{*xiaohanzhao, adelbert_chang, htzheng, ravenben*}*@cs.ucsb.edu, atish.dassarma@gmail.com*

ABSTRACT

Analysis of large graphs is critical to the ongoing growth of search engines and social networks. One class of queries centers around node affinity, often quantified by random-walk distances between node pairs, including *hitting time*, *commute time*, and *personalized PageRank* (PPR). Despite the potential of these "metrics," they are rarely, if ever, used in practice, largely due to extremely high computational costs.

In this paper, we investigate methods to scalably and efficiently compute random-walk distances, by "embedding" graphs and distances into points and distances in geometric coordinate spaces. We show that while existing graph coordinate systems (GCS) can accurately estimate shortest path distances, they produce significant errors when embedding random-walk distances. Based on our observations, we propose a new graph embedding system that explicitly accounts for per-node graph properties that affect random walk. Extensive experiments on a range of graphs show that our new approach can accurately estimate both symmetric and asymmetric random-walk distances. Once a graph is embedded, our system can answer queries between any two nodes in 8 microseconds, orders of magnitude faster than existing methods. Finally, we show that our system produces estimates that can replace ground truth in applications with minimal impact on application output.

1. INTRODUCTION

Analysis of large graphs is critical to the ongoing growth of search engines and social networks. One important measure in such analysis is a measure of proximity between pages on a web graph, or between people in a social network. Such measures can be quantified either by simple node distance metrics, *i.e.* shortest path, or by a more complete property of *affinity*, which is often quantified by random-walk distances such as commute time, hitting time and personalized PageRank.

There are numerous potential applications for these distances. For example, social networks like LinkedIn often provide a measure of similarity between a user and the owner of a profile she is reading. Pure graph distance is not truly reflective of the strength of ties between two users. An alternative uses the number of paths connecting them, *e.g.* counting the number of common neighbors. But this does not capture the impact of a user's node degree, *i.e.* m mutual friends between A and B is more significant when A and B each have few friends. Similar examples exist in the web, *i.e.* determining the similarity of a web search result with an ad-result produced by the same query. Finally, when performing semantic text analysis, determining contextual meaning often requires computing the relationship between specific keywords. By locating words in a wordnet graph, we can apply a robust measure of graph affinity to find the answer.

The immense power of random walks arises from the fact that it combines two very well studied simple notions of affinity between graph nodes, namely the graph distance, and the number of paths. Intuitively, two nodes are *similar* if they are close by in terms of distance. Independently, two nodes are more similar if they contain several paths between them. Random walk measures such as hitting time, commute time, or personalized PageRank beautifully capture both of these notions through the simple iterative random walk process.

Note, however, that computing any of these quantities is a very computationally expensive process. For example, hitting time is the expected number of hops in a random walk from a source node before it reaches a destination node. Given its inherent randomness, obtaining an expected hitting time from A to B requires computing hundreds or thousands of random walks, depending on the density of the underlying graph. Such costs are obviously intractable for today's massive graphs that have millions of nodes and billions of edges. Assuming the availability of sufficient memory resources, it takes on the order of minutes to compute shortest path via a single breadth-first-search (BFS). Computing a single hitting time on a massive graph can take anywhere from minutes to an hour or longer. Thus it is unsurprising that random-walk distances are rarely used in practice.

One approach to addressing the cost of these random-walk distances is approximation. These provide an attractive cost-benefit tradeoff for random walk distances, since the ground truth itself is highly variable. A number of approaches have been proposed to estimate personalized PageRank (PPR), including methods that use linear algebraic optimization [18, 8, 38] or Monte Carlo approximations [13, 3]. Another approach is to estimate the expected values using matrix computations [40, 6], but those methods can range from $O(n^3)$ to $O(\log n)$, and are limited to symmetric metrics, *i.e.* commute time, but not hitting time or personalized PageRank.

More recently, researchers have proposed the use of embedding graphs into geometric spaces as an alternative approach to capturing and estimating graph distances in near-constant time [47, 48]. These "graph coordinate systems" can accurately estimate node distances in large undirected graphs, with per-query latency (after

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at The 39th International Conference on Very Large Data Bases, August 26th - 30th 2013, Riva del Garda, Trento, Italy.
Proceedings of the VLDB Endowment, Vol. 6, No. 14
Copyright 2013 VLDB Endowment 2150-8097/13/14... $ 10.00.

initial pre-processing phase) 5 to 6 orders of magnitude lower than deterministic alternatives. As an alternative to graphs, geometric embeddings have the potential to capture and resolve queries for a variety of distance metrics in large graphs [14, 46].

In this paper, we explore the possibility of using a geometric space embedding to provide an efficient way to answer queries on random-walk distances, including commute time, hitting time and personalized PageRank. We begin by embedding these distances on a basic graph coordinate system and evaluating its accuracy. Not surprisingly, a traditional embedding system produces significant errors when estimating asymmetric distances. Based on our observations, we design a new graph coordinate system that explicitly accounts for asymmetry in random walks by capturing an intuition of graph density on a per-node basis. Extensive experiments on a variety of social graphs show that our embedding can not only embed asymmetric distance with very high accuracy, but also significantly improve the accuracy of symmetric metrics. In addition, we study the extremely high cost of computing the small sample of ground truth necessary for the embedding, and propose and evaluate simple techniques that generate ground truth samples with low computation cost. Finally, we use a small sample of application level tests to show that our embeddings enable (previously intractable) applications based on random walks, and these applications produce results with very small deviations from results using ground truth.

To the best of our knowledge, we provide the first general solution to enable the fast computation of graph distance functions based on random walks. We believe that our results are impactful for two reasons. First, our work makes a number of previously computationally intractable applications feasible. Second, by returning results to these queries in near real-time, we encourage researchers to further unlock their potential, by discovering new applications relying on them as basic primitives.

2. BACKGROUND AND CONTEXT

The goal of our work is to design a scalable system that quickly computes random-walk distances between any two nodes in undirected unweighted graphs. As a relatively new application of embedding to graphs, *graph coordinate systems* (GCS) have shown promise in estimating node distances and shortest paths in near-constant time, while demonstrating relatively high accuracy in empirical tests. Our hope is to expand on the general approach of embedding to geometric coordinates, but to do so in the context of both symmetric and asymmetric random-walk distances.

In this section, we first define in detail random-walk distances in undirected unweighted graphs, including hitting time, commute time and personalized PageRank. We then give background on the concept of graph coordinate systems, and identify the challenges that arise when this approach is applied to random-walk distances.

2.1 Random-Walk Based Distances

In undirected unweighted graphs, a random walk is a sequence of random steps. Consider an undirected unweighted graph G, with vertices V and edges E. Starting from node v_0, a random walk in G chooses its next destination: if we are at node v_k at the k^{th} step, we randomly select a neighbor v_{k+1} of v_k with probability $1/d(v_k)$ as the destination of the $(k+1)^{th}$ step, where $d(v_k)$ is the degree of node v_k. Thus, the sequence of random nodes $v_k (k = 0, 1, 2, \cdots)$ is a random walk from node v_0 in Graph G.

There are a number of distance measures based on random walks. Our work focuses on the three most popular random-walk distances, *Hitting Time*, *Commute Time* and *Personalized PageRank (PPR)*.

Hitting Time. Hitting time from node i to node j is the expected number of hops in a random walk starting from node i before it reaches node j for the first time. Since graph density and local structure around nodes i and j are different, hitting time from node i to node j is likely different from the hitting time from node j to node i. In other words, hitting time is asymmetric.

Commute Time. Commute time between two nodes i and j is the expected number of random walk hops from node i to j and then back to node i. Thus commute time is the sum of two hitting time distances, one from i to j and one from j to i. Thus, commute time is symmetric.

Personalized PageRank. Personalized PageRank (PPR) [21, 18] from node i to node j is the likelihood that a random walk starting from node i ends at node j with the reset probability α. In a random walk with reset, the reset probability α is the probability that at each hop, a node v can choose to selects itself as its next random walk step (*i.e.* resets its walk). In each step at node v_k, the random walk selects the current node v_k as the next step with reset probability α, and uniformly selects one of its neighbors with probability $1 - \alpha$. By starting m such random walks originating at node i, we count the number of random walks ending at node j on the T^{th} step (m_j), where T is a parameter chosen to capture the number of hops before the random walk probability converges for any given destination. PPR from node i to j is the ratio m_j/m.

All three distance measures have been used extensively in different contexts. Despite the simplicity, these measures are very powerful because they are able to incorporate two fundamental properties behind the affinity of pairs of nodes in graphs - specifically the graph distance, as well as the number of paths. In particular, the similarity between two nodes based on any of the three aforementioned random walk measures is likely to be higher if the two nodes are *closer* in the graph distance sense. Alternatively, given the same graph distance between two nodes, at a very high level, the random-walk based similarities are likely to be higher when *multiple paths* exist between these nodes. From an application standpoint, notice that any path between two nodes is a weak signal of similarity, and with multiple paths, any such reasonable notion of similarity should be reinforced. Similarly, a short path means the two nodes are close/similar.

2.2 Graph Coordinate Systems

Graph coordinate systems (GCS) [47, 48] are designed to accurately and efficiently estimate shortest path distances between node pairs in large graphs. They capture complex graph structure by embedding node distance relationships into a geometric space, *e.g.* Euclidean space in Orion [47] and Hyperbolic space in Rigel [48]. Each node is represented by a set of coordinates such that geometric distances between any nodes can preserve their real shortest path lengths in the actual graph. For example in Figure 1, the shortest path between node A and E is 3 in the graph and the Euclidean distance calculated from their embedded coordinates is 2.9.

Before answering queries, graph coordinate systems must first embed the graph into a geometric space. This is done via a centralized landmark approach, where a relatively small number ($L(\ll n)$) of landmark nodes are first embedded. Using a global optimization algorithm, the system calculates coordinates of landmark nodes in the geometric space by matching their geometric distances to their real pair-wise shortest path distances. Once landmarks are properly embedded, each graph node is embedded in the geometric space so that its geometric distances to a subset of landmarks, $k(< L)$ landmarks, are as close as possible to its real shortest path to those landmarks in the graph. [47, 48] apply Simplex Downhill algorithm [30] to optimize node coordinates. After all nodes

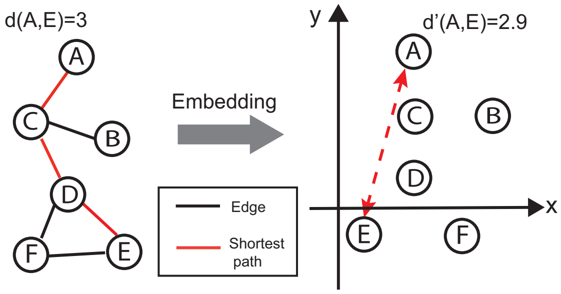


Figure 1: An example to map graph into a Euclidean space. The shortest path between node A and E is 3 hops in the left graph and the estimated Euclidean distance between them is 2.9 hops in the right graph.

are embedded into the space, distances between any pair of nodes can be approximated in constant time by computing their geometric distances.

The landmark approach is ideal for graph coordinate systems because it minimizes the number of Breadth-first-search (BFS) operations necessary. A graph embedding using a pairwise or decentralized scheme would require $O(n^2)$ computations, for a network of n nodes. In contrast, a landmark scheme with L landmarks only requires $O(L)$ BFS operations to measure distances between all nodes and landmarks. Landmark embedding can be further simplified by breaking the landmark nodes into two subsets, and using one subset as anchors for a larger second subset.

The strength of graph coordinate system comes from that once a graph is embedded, a GCS can resolve shortest path queries in constant time using the embedded coordinates, *i.e.* $O(1)$, independent of the size of the graph. [47, 48] show, on average, their system responses each query in microseconds (μs), which is orders of magnitude smaller than basic BFS. This is highly attractive for applications like network centrality computation or social-distance based ranking, each of which relies on resolving a large number of shortest path distance queries. Rigel [48] also shows that the initial graph embedding process can scale by distributing the preprocessing across distributed machines with near linear speedup.

2.3 Prior Art and Challenges

There are two classes of existing techniques to compute random-walk based distances. The first class uses linear algebraic techniques, such as computing pseudo-inverse of the Laplacian matrix of the graph for commute time [36] and Power Iteration [21] for PageRank. The second class of approaches is simulating random walk via a Monte Carlo method [13, 3]. The basic idea is to directly emulate random walks on a graph and repeat random walks for enough iterations to determine the expected distance.

However, both approaches are still quite time consuming. Take commute time for example. In a graph of n nodes, standard inverse matrix computation for commute time requires $O(n^3)$ time, which falls far short of scaling to large graphs. The Monte Carlo method is also intractable, and cannot scale with graph size. Our own experiments show that it can take more than 15 minutes to compute commute time between a single pair of nodes in a graph of $250K$ nodes. While numerous optimizations have been proposed to improve these techniques, it is clear that providing fast answers to queries on million node graphs will require a dramatically different approach.

Embedding Random-Walk Distances. Our goal is to test the feasibility of using geometric space embeddings to capture random-walk distances. Specifically, we look for a graph coordinate system that maps nodes in a graph into a geometric space of fixed dimension, where distances between nodes represent estimated values of expected random-walk distances. Using a node's coordinate position in the space, we can accurately estimate the corresponding random-walk distances between any two nodes in constant time.

A naive solution is to apply the design of current graph coordinate systems, and substitute random-walk based distances for shortest path distances. However, two key properties of random-walk based distances pose real challenges and prevent us from using this naive approach. We summarize these two issues below, and further explore them in Section 3.

Asymmetry. The first and most critical difference between random-walk distances and shortest path distances is symmetry. In undirected graphs, shortest path length is symmetric by definition. In contrast, hitting time and PPR are asymmetric [26, 18], *i.e.* distance (either hitting time or PPR) from node A to B may not be the same as the distance in the reverse direction. In addition, distances in any geometric space, *e.g.* Euclidean space or Hyperbolic space, are symmetric. This leads us to believe that embedding random-walk distances on coordinate spaces will produce significant errors.

Cost of precomputation. Second, we note that it takes significantly more time to obtain ground truth of random-walk based distances, especially for hitting time and commute time. Depending on graph structure, arriving at a stable expected value for random walks can require thousands of independent random walks. For instance, using a commodity server with sufficient main-memory, computing expected hitting time from one landmark node to all nodes in a $250K$-node social network graph takes 60 days. In contrast, it takes only 2 hours to compute the shortest path distance (using BFS) between 100 landmarks and all nodes in the graph. Therefore, to make any embedding system practical for random-walk distances, we also need to address the issue of efficiently obtaining ground truth. We address this issue further in Section 4.3.

3. A NAIVE RANDOM WALK EMBEDDING

We begin by examining whether we can simply apply existing graph coordinate systems to embed random-walk based distances. Using 10 real graphs and 1 synthetic planar graph, we have examined the embedding accuracy of two existing graph coordinate systems, Orion [47] and Rigel [48]. These results shed light on what changes we need to make to build a more accurate system for estimating random-walk distances. Our results show that Orion, a Euclidean space coordinate system, consistently outperforms Rigel, a Hyperbolic space coordinate system.[1] Therefore, we limit ourselves to tests on Orion in the rest of this paper.

3.1 Experiment Setup

We implemented Orion to embed random-walk based distances into a Euclidean space. Like [47], we select $L = 100$ random nodes as landmarks[2], and randomly choose $l = 16$ of these landmarks as the initial set. For each node, we use $k = 16$ landmarks to compute their coordinates. By default, we use coordinates of 10 dimensions, because further increasing the dimension size offers little improvement in embedding accuracy but introduces extra computation overhead.

[1] The Hyperbolic space is good at capturing distance metrics with small variance, *e.g.* shortest path, but loses accuracy when modeling high-variance metrics like hitting time and commute time.

[2] While Orion proposed three choices of landmark selection (descending node degree, descending node degree with separation, and random), our experiments show that randomly selecting landmarks achieves the highest accuracy.

Ground Truth Computation. One challenging component of the embedding process is computing ground truth values of the random-walk distances between landmarks and regular nodes. Our solution in these experiments is brute force search, where we simulate multi-round random walks on each social graph and derive the mean values. Because computing ground truth for all node pairs is extremely costly[3], we randomly select 2000 nodes to embed (after embedding all the landmark nodes).

Hitting time computation. The hitting time from node i to j, $H(i, j)$, is the expected number of random walk hops from i to j. To compute $H(i, j)$, we simulate a random walk starting from i until it reaches j for the first time, repeat the process N times, and compute the average of the hop count from each walk. We choose $N = 2000$ because our experiments show that the average hop count stabilizes at this value, for all graphs in Table 1.

Commute time computation. Once we measure the hitting time from node i to node j and the hitting time in the other direction, *i.e.* $H(i, j)$ and $H(j, i)$, we can easily derive the commute time between node i and j, $C(i, j) = H(i, j) + H(j, i)$.

PPR computation. We initiate a random walk from node i with reset probability α, terminate the walk at the T^th step, and repeat the process for N times. We then compute m_j, the number of times that a node j is visited at the T^th step across all N rounds. The PPR from node i to j is computed as $PPR(i, j) = m_j/N$. Our experiments use $\alpha = 0.15$, the common choice of PPR computations [18, 5], and $T = \log n/\alpha$ because prior work proved that PageRank converges in $O(\log n/\alpha)$ hops [11]. We also found that $N = 8000$ is adequate to get a stable PPR estimation.

Unlike hitting time and commute time, PPR cannot be directly embedded using graph coordinates. This is because of two reasons. First, the embedding process assumes when two nodes are close to each other in the embedded graph, their actual distance is also small. This is true for shortest path, hitting time and commute time, but not for PPR. The larger the PPR value, the more similar (and thus closer) the two nodes. Second, the value of PPR is always between 0 and 1, a range that cannot be accurately captured by Orion. We address these two issues by embedding an alternative metric $(1 - PPR(i, j)) \cdot 10^6$ instead of $PPR(i, j)$ itself.

Datasets and Accuracy Measures. Using 10 real graphs and 1 synthetic graph, we evaluate the accuracy of random-walk distances estimated by Orion. Listed in Table 1, Monterey Bay, Santa Barbara and Egypt graphs are social graphs collected from three regional networks of Facebook in 2008 [41], with sizes ranging from 6K nodes and 31K edges to 246K nodes and 1M edges. The next 7 graphs are from various networks, including a collaboration network graph from arXiv [24], an Internet autonomous systems (AS) graph from CAIDA [23], a citation graph from arXiv [23], a snapshot of the Gnutella peer-to-peer file sharing network [23], a measured Email network graph of a large European research institution [24], an Amazon product co-purchasing graph [22], and a web graph from Notre Dame [1]. Finally, as a representative of a planar graph, we add a synthetic graph generated by the Dorogovtsev-Goltsev-Mendes Internet model [12]. We choose these graphs because they can demonstrate the scalability and applicability of graph coordinate systems across a variety of graph topologies.

We evaluate the accuracy of random-walk distance estimation using the notion of *relative error*, the same one used by [47]. For each node pair, the relative error is the absolute difference between the geometric node distance (*i.e.* the estimated random-walk distance) and the ground truth random-walk distance, normalized by the ground truth value. A relative error of 0 means the estimated random-walk distance matches the ground truth value perfectly. Because computing ground truth for all node pairs is extremely costly and time-consuming, we randomly sample 1000 node pairs out of the 2000 embedded nodes, and examine their random-walk distance estimations.

Networks	# of Nodes	# of Edges	Avg. Degree
MontereyBay	6,115	31,374	10.26
Santa Barbara	26,566	226,566	17.05
Egypt	246,692	1,618,085	13.12
Collaboration	21,363	91,342	8.55
AS	26,475	533,831	40.33
Citation	34,401	420,828	24.47
P2P	62,562	147,878	4.73
Email	224,832	339,925	3.02
Amazon	262,111	899,792	6.87
Web	325,729	1,117,563	6.86
Planar	265,722	531,441	3.99

Table 1: Datasets used in our experiments.

3.2 Accuracy Results

Commute Time. Figure 2(a) shows the CDF of relative error when we embed commute time using Orion in all three social graphs. For all three graphs, the relative error is low and follows a similar distribution. For Santa Barbara and Monterey Bay, 90% of node pairs achieve less than 0.3 relative error, and for Egypt, the value increases slightly to 0.4. The results measured on the two sparse graphs are shown in Figure 3(a), similar to the results in social networks. These are similar to the accuracy result of shortest path embedding [47].

Hitting Time. The accuracy of the embedded hitting time in social networks is shown in Figure 2(b). Again the results for all three social graphs are similar. But compared to commute time, the relative error is significantly larger. The majority of node pairs experience at least 0.7 relative error, and only 15% of node pairs have relative error lower than 0.3. Similarly, we find the accuracy of hitting time embedding in two sparse graphs are much worse than their commute time embedding, which is shown in Figure 3(b).

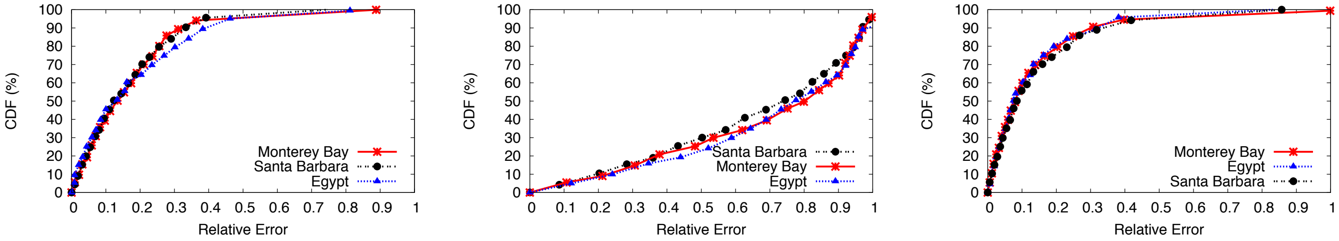
PPR Embedding. The performance of PPR embedding in social graphs, shown in Figure 2(c), is similar to that of commute time. 90% of node pairs experience 0.35 and less relative error. Again, the two sparse graphs produce almost the same results in Figure 3(b).

3.3 Key Insights

In the following, we discuss the key insights observed from our experiments. Our first observation is that Orion, while originally designed to handle shortest path, can also embed commute time at a reasonable accuracy. This is because like shortest path, commute time is a symmetric metric, *i.e.* the distance from node i to j equals to the distance from j to i. Such symmetric values can be properly captured by Euclidean distances. We observe that there is long tail in the relative error CDF of commute time embedding. This is caused by low degree nodes, because Orion can not accurately position low degree nodes that have poor connections to the majority of graph.

Our second observation is that hitting time cannot be properly embedded by Orion because of its asymmetry. Specifically, if the local graph structure around a node i is very different from that

[3] Computing the ground truth random walk between a single landmark and all non-landmark nodes takes up to 60 days on a commodity server.

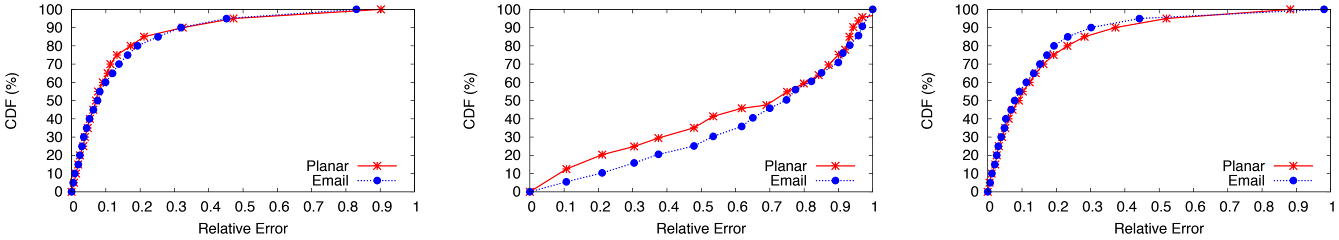


(a) Commute Time

(b) Hitting Time

(c) PPR

Figure 2: Relative error CDF of random-walk distances embedding using Orion in social graphs



(a) Commute Time

(b) Hitting Time

(c) PPR

Figure 3: Relative error CDF of random-walk distances embedding using Orion in sparse graphs

of a node j, *e.g.* i connects to a large group of local nodes while j only has a few neighbors, the hitting time from i to j will differ significantly from the hitting time from j to i. To quantify the degree of such asymmetry, we compute for each node pair (i, j), the relative difference of the hitting times in both directions: $\gamma = \frac{|H(i,j)-H(j,i)|}{max(H(i,j),H(j,i))}$. The CDF of the γ value is shown in Figure 4 for the 1000 randomly sampled node pairs. The majority of the node pairs (70%+) have $\gamma > 0.5$. This confirms that hitting time is highly asymmetric, and thus cannot be properly captured by Orion's Euclidean distance measurement that is symmetric.

Finally, an interesting observation is that while PPR is also an asymmetric metric, its embedding error is similar to that of commute time and much better than that of hitting time. This is because an inherent artifact of the PPR computations. Our experiments show that the majority of PPR values are zeros. In particular, for the three social graphs tested, 63.6% node pairs have zero PPR values in both directions, while for the rest, the degree of asymmetry is less than 0.006. This means that PPR essentially becomes a symmetric metric, and explains why the embedding performance is closer to that of commute time.

Summary of Observations. As an initial effort, we apply Orion, a graph coordinate system developed for shortest path estimation, to embed three random-walk distances (commute time, hitting time and PPR). Experiments on three real social graphs show that Orion embeds commute time and PPR at a reasonable accuracy similar to that of shortest path, but produces large errors on asymmetric distances like hitting time. This motivates us to search for a new graph coordinate system that can properly embed both symmetric and asymmetric distances.

4. A NEW GRAPH COORDINATE SYSTEM

Our experiments in Section 3 show that a traditional embedding system produces significant errors when estimating random walk distances, especially asymmetric distances. In this section, we present *Leo*, a new graph coordinate system that explicitly accounts for asymmetry in random walks. The intuition behind our design is that asymmetry in random walks is caused by distinct "local" graph density around each node, and by capturing such effect on a per-node basis, one can effectively model random walks via graph coordinates.

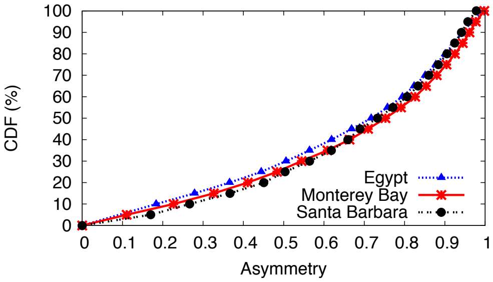


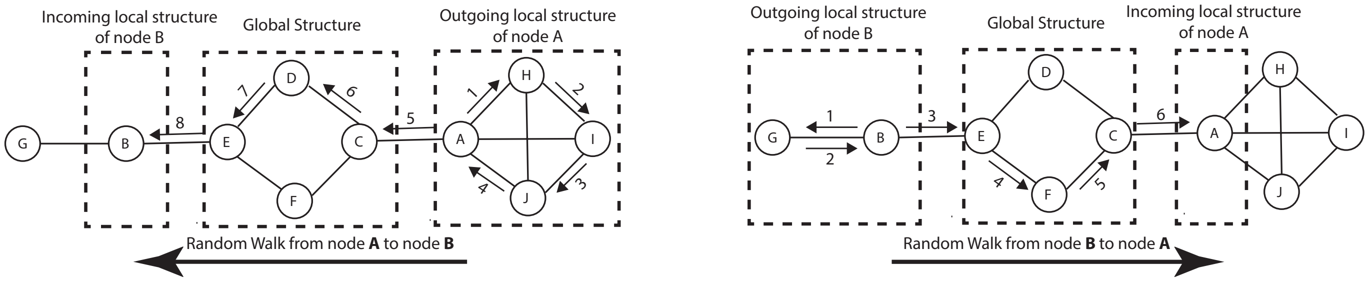
Figure 4: CDF of the hitting time asymmetry, quantified by the normalized difference between hitting times in two directions: $\gamma = \frac{|H(i,j)-H(j,i)|}{max(H(i,j),H(j,i))}$. More than 70% sampled node pairs have more than 0.5 normalized difference, indicating that hitting time is an asymmetric measure.

Our discussion on Leo begins by analyzing the cause of asymmetry in hitting time based random walks, where we illustrate the significant effect of local graph density. We then present a new coordinate space combining Euclidean coordinates and heights to capture such effect on a per-node basis, followed by a description of the overall embedding process.

4.1 A Closer Look at Hitting Time

To illustrate the cause of asymmetry in hitting times, we consider a toy example using random walks between node A and B in Figure 5. In this example, an arrow from node i to node j represents a random walk step from i to j, and the number k on top of the arrow represents the sequential order of the current random walk, *i.e.* the k^{th} step.

Figure 5(a) shows an instance of random walk from node A to


Incoming local structure of node B
Global Structure
Outgoing local structure of node A
G
B
E
D
F
C
A
H
I
J
8
7
6
5
1
2
3
4
Random Walk from node **A** to node **B**
Outgoing local structure of node B
Global Structure
Incoming local structure of node A
G
B
E
D
F
C
A
H
I
J
1
2
3
4
5
6
Random Walk from node **B** to node **A**

(a) Random Walk from Node A to B

(b) Random Walk from Node B to A

Figure 5: Example of a random walk from node A to B in (a) and a random walk from node B to A in (b).

B. Since A's neighborhood is tightly connected, *i.e.* a clique consisting of A, H, I and J, it takes 5 hops to leave A's local structure and reach node C. The subsequent random walk takes 2 hops to reach node B's local neighborhood E, and another 1 hop to reach B. In total, the random walk takes 8 hops. Figure 5(b) illustrates the random walk from node B to A. Here node B only has two neighbors, and the current instance of random walk takes 3 hops to leave B's local neighborhood. It takes another 2 hops to reach C, and another extra hop to reach A. In total this random walk from B to A only requires 6 hops, 2 hops less than that from A to B.

This example also sheds light on one potential view of why random walks are asymmetric. We can think of random walks as traversing through three abstract "regions" of the graph: first exiting an *outgoing local structure* near the source node, moving across a *backbone global structure* in the graph, and finally finding the destination node inside its an *incoming local structure*. Our intuition into the asymmetry of hitting time is that random walk distances are largely dominated by a node's incoming and outgoing local structures, while traversal across the global graph structure can be thought of as fairly symmetric. For example, both node A and B's incoming (Figure 5(b)) and outgoing (Figure 5(a)) local structures are significantly different, leading to the large difference of 2 hops between their corresponding average random walk distances. Clearly, these differences cannot be captured by traditional embeddings.

4.2 Per-Node Height Vectors

The above intuition implies that we need a graph coordinate system with three components, two asymmetric components that capture each node's local structure for outgoing and incoming random walks, and a symmetric component that captures the global structure. Coordinate systems (*i.e.* Euclidean, Hyperbolic or Spherical spaces) can easily capture the symmetric component. Our task is to identify and model the remaining two directional asymmetric components. For this, we introduce *directional height vectors*.

Node Heights. The idea of an always present, per node cost to an embedding was introduced in Vivaldi [10], a network coordinate system that used a single per-node cost to capture congestion delays at the edge routers of the network. In our case, the new parameters represent an abstract view of local graph density near the source and destination nodes. More importantly, the directionality of our distances dictates that we use *two* distinct height vectors for each node: $h_{in}(i)$ and $h_{out}(i)$ ($h_{in}(i) \geq 0$, $h_{out}(i) \geq 0$). As we will show later, the values of these parameters can vary significantly based on the overall topological properties of the graph.

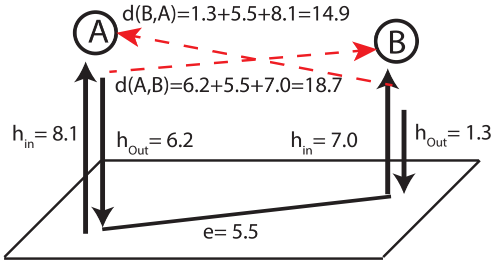


Figure 6: Two nodes in our new coordinate space composed of the 2D Euclidean space and two heights. The vertical lines represent height vectors, and the arrows mark the directionality (incoming/outgoing). The line e represents the distance in the Euclidean space, and the red dashes represent the predicted random walk distances produced by our system. Note that a node's outgoing vector is typically smaller than its incoming vector.

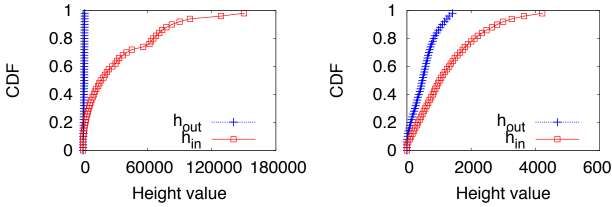
Our embedding system for random walks computes predicted distances by combining two appropriate height vectors with an undirected distance captured by the baseline embedding space. As shown in Figure 6, a random walk from node A to B will first exit its local structure with outgoing height $h_{out}(i)$, followed by the core global structure represented by an Euclidean distance between A and B, and finally through the local structure of j with incoming height $h_{in}(j)$. The total expected random walk length, *i.e.* predicted hitting time, is the sum of the Euclidean distance and two heights, $h_{out}(i)$ and $h_{in}(j)$:

$$d(A, B) = h_{out}(A) + \sqrt{\sum_{i=1}^{n}(x_i(A) - x_i(B))^2} + h_{in}(B) \quad (1)$$

where the vector $\{x_i\}_{i=1}^{n}$ represents the n-dimension Euclidean coordinates of node i.

The example in Figure 6 shows a basic 2-dimensional Euclidean plane coordinate space. The heights of node A are $h_{in} = 8.1$ and $h_{out} = 6.2$, and those for node B are $h_{in} = 7.0$ and $h_{out} = 1.3$. Their embedded distance in the 2D Euclidean plane is 5.5. We compute the random walk distance from node A to B (the top dash line in Figure 6), $d(A, B)$, by summing node A's h_{out}, the 2D Euclidean distance and node B's h_{in}, which is 18.7 in total. Similarly, the distance from node B to node A (the bottom dash line) is the sum of the Euclidean distance and node B's h_{out} and node A's h_{in}, producing a distance of 14.9.

Embedding Process. By treating the node heights as two extra components in the coordinate system, *Leo* uses an embedding process similar to prior systems. The main process is driven by optimizing the coordinate positions and height vectors to minimize distortion between the embedding and the ground truth of the graph.



(a) MontereyBay (b) Internet Topology

Figure 7: CDF of h_{in} and h_{out} of two graphs

The key change is that we must compute ground truth in terms of random walk distances, and use Equation (1) for node distance.

Space Usage. Leo is highly space efficient. Once a graph with n nodes has been embedded into d-dimension space, we only need to load the embedded graph coordinates into memory to give fast responses to queries. Thus, the space usage is $O(d \times n)$.

Understanding Height Vectors. Leo's direction-specific, per-node height vectors are key to accurately modeling distances in random walks. We dig deeper using two examples to better understand how and why they work for different graph topologies.

We first look at the results of embedding the Monterey Bay Facebook graph (Table 1). Figure 7(a) plots the CDF of $h_{in}(.)$ and $h_{out}(.)$ We see that h_{in} varies significantly across nodes, going as high as more than 150,000. Yet most values of h_{out} are 4-5 orders of magnitude smaller than h_{in}, with the large majority of them remaining at 0! Small h_{out} values capture the fact that in the MontereyBay social graph, each node's local structure is well-connected to the global structure, thus random walks from a node can quickly reach the global structure from many points. The extremely large h_{in} values show that social graphs like Monterey Bay also have well-connected local structure, where each random hop toward a specific destination node has a low probability of finding it. We believe these specific values are ideal for small-world graphs such as those from social networks.

To test our hypothesis, we introduce a second graph of similar size to Monterey Bay, but with dramatically different structural properties. We generate a synthetic Internet-like graph with 6000 nodes and 50,833 edges using the Transit-stub model [45]. Transit-stub networks have a two-level hierarchy: a random graph emulating the Internet backbone as the top level, and a set of well-connected random graphs each connecting to a single node in the backbone as the second level. Here, network diameters can be high, and the hierarchical topology is opposite of a small-world social graph. Figure 7(b) shows CDF of height vectors generated by embedding this graph. Here h_{out} and h_{in} are reasonably similar and within a factor of 2 of each other. This confirms our intuition: sparse connections between the backbone and each stub network increase the costs of finding the exit path to the backbone, *i.e.* large h_{out}); dense connections within each local structure (*i.e.* large h_{in}).

These examples show how decoupling incoming and outgoing height vectors allows Leo to successfully adapt to a wide range of graph structures. The result is a significantly more general model that will produce accurate results on a variety of graph topologies. This is the key difference between Leo and a system like Vivaldi.

4.3 Fast Precomputation

As we mentioned earlier, a critical challenge in embedding random walk distances is the cost of ground truth computation. When applying the Orion embedding process, the embedding must make $(2 \cdot L \cdot n) \cdot 2000$ pairwise random walks to measure actual hitting time (and commute time) between landmarks and non-landmark nodes. Here L and n are the number of landmarks and graph nodes, respectively. For a large graph like the Egypt Facebook graph, it takes around 60 days just to compute the ground truth distances between a single landmark node and all other nodes.

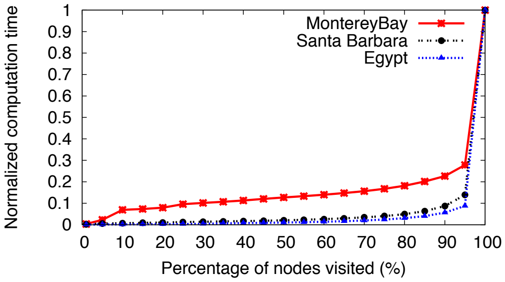


Figure 8: Percentage of visited nodes vs. the computation time of a random walk normalized by the time to visit all nodes.

To address this challenge, we propose a novel precomputation method that reduces the pairwise random walks to $(L + n) \cdot 2000$, reducing the total embedding time for Egypt from 60 to only 7 days.

Multi-destination Random Walk. The fast precomputation algorithm is based on a random walk with multiple destinations. Running such random walk from a node can produce random walk steps from this node to multiple nodes, which is similar to BFS algorithm. More specifically, a random walk starting from node i follows its definition to select its next step. If the random walk visits a node j for the first time, we record the current walk steps as one trial for hitting time measurement from node i to node j. Instead of stopping this random walk as defined hitting time, the random walk continues and records its current steps when it reaches a new node for the first time. This random walk can stop when it visits require number of k nodes or its steps get to the maximum steps. As a result, one such random walk can measure the steps to multiple nodes when they are visited for the first time.

Although such random walk with multiple destinations can reduce the number of random walks a lot, it is still not scalable to large graphs if it stops when it visits all graph nodes. Take Egypt as an example again. One such random walk takes 1.5 minutes from a node to reach each node in the graph. To get the converged hitting time from a node, we need to repeat such random walk for 2000 times, which takes 48 hours. To understand this efficiency of random walk, we plot the percentage of nodes visited by the random walk vs. the computation time normalized by the time visiting all nodes in Figure 8. We find that 90% of the nodes can be visited within 10% $\sim$ 20% of the normalized computation time. That means in a random walk visiting all nodes, more than 80% of time is used to visit less than 10% of nodes, which is the main reason causing high computation cost. Thus, we explore a tradeoff between efficiency and quantity of visited nodes and find that it is a better compromise when a random walk visits 90% of nodes.

We run the random walk starting from a node for more times to make up the 10% not visited nodes. Recall that we measure stable hitting time from node i to node j by repeating the random walk from node i to j for 2000 times. Similarly, we have to repeat the random walk with multiple destinations for several times for a reasonable expectation. In addition, since such random walks have no explicit destinations, we can not promise each visited node can be visited for 2000 times, which can provide a stable hitting time, after 2000 times repeating random walks. In other words, we may need to run such random walks for more than 2000 times.

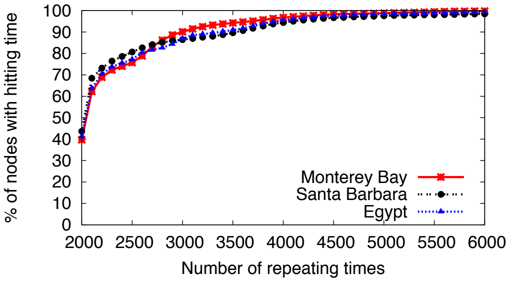


Figure 9: Percentage of nodes with stable hitting time vs. Repeating times of random walks

We empirically repeat the random walk starting from a node for N times, where N is from 2000 times to 6000 times. In Figure 9, we show the percentage of nodes which are visited for at least 2000 times by random walks when we repeat random walks for different times. We find that when we repeat random walks for more times, the percentage of nodes with stable hitting time increases. Specifically, when we repeat the random walk for 2000 times, only 40% of the nodes get stable hitting time. When the repeating time is 6000 times, 99% of the nodes have stable hitting time. Thus, starting from a node, we repeat the random walks for 6000 times to get more nodes with stable hitting time.

For the remaining 1% nodes without stable hitting time, we run simple end-to-end random walks from the source node to it to ensure that they are visited for 2000 times. Finally, we can compute the ground truth of hitting time from one node to all the other nodes in the graph more efficiently.

In one word, this optimized algorithm works based on random walks with a soft cutoff. That is, a random walk starting from node i records the steps to each node that is visited for the first time and stops when it visits 90% of nodes in the graph. To get stable hitting time for each visited node, we repeat such random walk for 6000 times. For the remaining nodes which are visited for only $k(\leq 2000)$ times, we carry out end-to-end random walks from node i to it for $2000 - k$ times to guarantee that it has stable hitting time.

We can directly run this algorithm for each landmark to fast compute the actual hitting time from the landmark to all the nodes. To compute the ground truth from non-landmarks to landmarks, we can use this algorithm for each non-landmark with a small modification. In detail, since each node only needs to compute the hitting time to a subset of landmarks, *i.e.* 16 landmarks in our paper, the random walk starting from it stops when 16 landmarks are reached. As a whole, we can efficiently measure the ground truth of hitting time and commute time for the embedding process using this fast computation algorithm.

5. PERFORMANCE EVALUATION

In this section, we first understand the performance of Leo in term of accuracy and speed. Specifically, we investigate the accuracy of random-walk distance estimation using Leo compared against Orion and study the impact of number of dimensions on its estimation accuracy. Then, we measure its efficiency by using average response time for pairwise queries. Second, we examine the utility of this system in two important applications built on random-walk distances, *i.e* search ranking and link prediction.

5.1 Accuracy

Compared to Orion. We examine the accuracy of hitting time, commute time and PPR embedding on the graphs in Table 1. To make fair comparison to the accuracy of 10-dimension Orion in section 3, we use Leo to embed all three distances into a space of 10-dimension Euclidean coordinates plus 2 heights. For simplicity, we call the space of d-dimension Euclidean coordinates plus 2 heights in Leo as a d-dimension Leo. Since hitting time computation is intractable, we sample 1000 random pairs of nodes from each graph and measure the actual hitting time, commute time and PPR between them for comparison. To avoid possible impact of landmarks on results, we choose the 1000 node pairs randomly from all non-landmark nodes. We use two metrics to quantify the accuracy of the embedding system. One is relative error, and the other is the 90^{th} percentile relative error over all nodes pairs (90% *relative error*). Since Leo performs consistently better than Orion, we focus on the CDF of relative error in Egypt and show the 90% relative error of all graphs.

Figure 10 shows CDF of relative error of hitting time estimation using Leo compared to the results from Orion. We find that Leo can significantly improve the estimation accuracy for hitting time. Specifically, for 90% of node pairs, the relative error is less than 0.1 using Leo while the relative error of Orion is more than 0.95. In other word, the accuracy improvement of Leo is 90%.

We plot CDF of relative error of PPR estimation in Figure 11. Similar to results in Figure 10, it shows that Leo is more accurate in estimating PPR. For example, for 90% of node pairs in Figure 11, the relative error of Leo is 0.005. This is much smaller than the relative error of Orion, *i.e.* $0.3 \sim 0.4$ for 90% pairs of nodes. The accuracy in estimating PPR is improved 98% by Leo. Both the results in Figure 10 and 11 show that the two heights introduced in Leo can accurately capture the asymmetric random-walk distances.

We also use Leo to embed symmetric commute time and compare its accuracy to the results of Orion in Figure 12. It shows that Leo significantly outperforms Orion that was designed for symmetric distances. Still for 90% pairs of nodes, Leo produces relative error 0.2 while the error in Orion is 0.38. Our measurement shows that low degree nodes tend to have large heights while high degree nodes tend to have small heights. This means that our heights can help to capture the local structure around nodes which have poor connection to the core of the graph. Thus, Leo can also capture symmetric distances more accurately than Orion.

We show the 90% relative error of 10-dimension Leo and 10-dimension Orion over all graphs in Table 2. Leo is consistently more accurate than Orion across all graphs and all metrics. For hitting time, the accuracy is improved by $67\% - 94\%$ by Leo. Among all graphs, we notice that the 90% relative errors of three sparse graphs, *i.e.* Planar, Email and Amazon, are slightly higher. Their much lower density means random walks pay a lower price both exiting their local cliques and trying to find their destination nodes. Not surprisingly, our results show that the symmetric global structures make up a much bigger component of the total random walk distance in these graphs. The higher relative errors likely come from estimation errors in the symmetric global distances. For PPR, we find that Leo improves the accuracy $98\% - 99\%$ for all graphs. For symmetric commute time, Leo also produces consistently better accuracy for all graphs, by $25\% - 44\%$ compared to Orion.

Impact of dimensionality. We also study how the number of dimensions used in the embedding impacts the accuracy of random-walk distance estimation. We vary the dimensions of the Euclidean coordinates from 0 to 10. A 0-dimension space in our system means that no Euclidean coordinates are used for embedding the global structure component, and we only use two heights to represent a node's local structure. Since it is symmetric, the accuracy of commute time embedding increases as the embedding dimension increases. In addition, the number of dimensions has no sig-

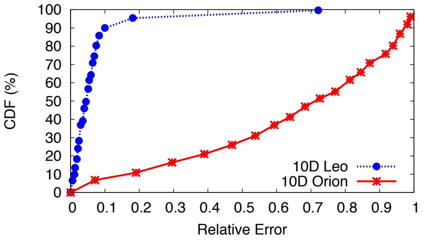


Figure 10: Relative error CDF of hitting time embedding in Egypt using 10D Leo vs. 10D Orion.

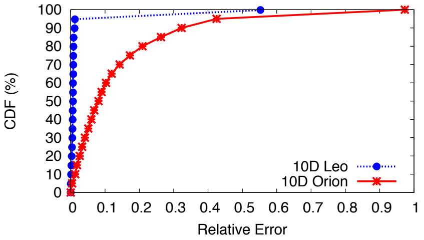


Figure 11: Relative error CDF of PPR embedding in Egypt using 10D Leo vs. Orion.

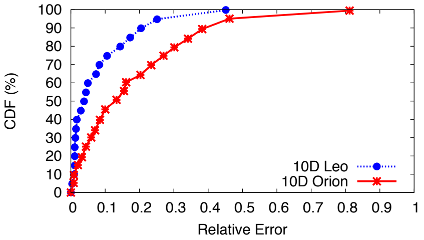


Figure 12: Relative error CDF of Commute Time embedding in Egypt using 10D Leo vs. 10D Orion.

Metric	System	MontereyBay	SB	Egypt	Collab	AS	Citation	P2P	Email	Amazon	Web	Planar
Hitting Time	Orion	0.970	0.919	0.977	0.988	0.971	0.962	0.914	0.987	0.945	0.956	0.969
	Leo	0.119	0.058	0.100	0.121	0.084	0.058	0.100	0.161	0.238	0.058	0.322
PPR	Orion	0.307	0.318	0.323	0.314	0.331	0.300	0.342	0.351	0.342	0.322	0.354
	Leo	0.004	0.006	0.003	0.004	0.004	0.005	0.006	0.003	0.004	0.005	0.006
Commute Time	Orion	0.311	0.333	0.383	0.352	0.325	0.341	0.301	0.311	0.362	0.381	0.375
	Leo	0.211	0.244	0.214	0.242	0.232	0.199	0.205	0.211	0.223	0.214	0.283

Table 2: 90th percentile relative errors for Hitting time, PPR and Commute time (Leo vs. Orion w/ 10 dimensions)

nificant impact on PPR accuracy. Thus we omit those plots for brevity and instead focus on the hitting time results. For clarity, we show the embedding accuracy using 0-dimension Leo and 10-dimension Leo. Figure 13(a) shows the results of hitting time in Egypt to demonstrate the impact. We find that the accuracy of 0-dimension Leo embedding is similar to than 10-dimension Leo embedding. This indicates that the asymmetric local structure of Egypt graph dominates the total random walk distance. In other words, two heights are enough to accurately capture the hitting time in the Egypt graph. In fact, we found this to be consistently true for our small-world and high density graphs, including the other two social graphs.

However, the results in our sparser, more hierarchical graphs look quite different. These include the Planar, Email and Amazon graphs. Figure 13(b) shows that 10-dimension Leo embedding of planar graph is more accurate than its 0-dimension Leo embedding. For example, for 90% nodes, the relative error in 10-dimension embedding space is 0.3, which is half of the error in 0-dimension space. We observe the same trend in the other three graphs and show the results of Email graph in Figure 13(c). Both results show that the symmetric component in the design of Leo, *i.e.* the Euclidean coordinate in Equation 1, is necessary, especially as the symmetric global structure of network increases. Again, this validates our hypothesis that the less dense a network is, the lower the cost of exiting local subgraphs and finding destination nodes. Thus the relative cost of our directional height vectors decreases, and the symmetric component grows in importance.

5.2 Embedding and Query Performance

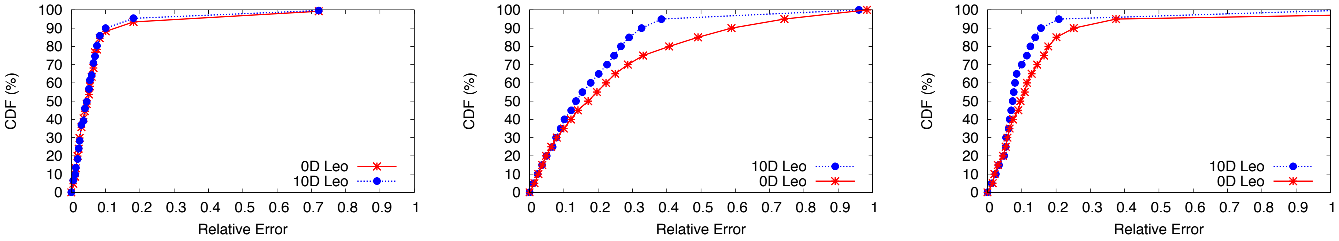
We study the efficiency of our embedding system in this section, including up-front bootstrap costs and average response time for a query. To evaluate the bootstrap time, we measure results for both a single thread instance and a distributed version parallelized across 100 servers. All experiments are measured on a 2Ghz, 8-core Intel Xeon machine with 192GB RAM, and all graphs are embedded into a 10-dimension Leo using 2 height vectors per-node. We show computation time on the largest three real graphs in Table 1, *i.e.* Egypt, Amazon and Web graphs.

The bootstrap process of Leo includes two phases. The first phase is the precomputation phase, which is to compute the actual distances between landmarks and non-landmarks. We apply our proposed fast precomputation algorithm to compute hitting time and commute time in this phase. The second phase is to embed the random walk distances into a low-dimension space. We measure the computation time of the precomputation phase and embedding the graph into a 10-dimension space using one single thread. To minimize the bootstrap time, we then parallelize the bootstrap across 100 servers and use the longest computation time of the 100 servers as the parallel bootstrap time.

We show the bootstrap time for hitting time, commute time and PPR using one single thread in Table 3. Since the commute time between node i and node j is the sum of hitting time from node i to j and hitting time for the reverse direction, the computation time of commute time between landmarks and non landmarks is equal to the time to compute the ground truth of hitting time. Since the maximum random walk hops for PPR is $log(n)/\alpha$, much smaller than the network size, its computation time is much faster than hitting time. For each graph, we find that the majority of bootstrap time is used to measure the ground truth between landmarks and non-landmarks. For example, to embed hitting time in Egypt, the precomputation for hitting time requires 168 hours while the embedding time is around 2 hours using one single thread.

We parallelize the bootstrap across 100 servers to reduce the bootstrap time, and results are shown in Table 3. Since the precomputation process is embarrassingly parallel, we do achieve very high speedups (90x for hitting time and commute time, and 70x for PPR). Since landmarks can only be embedded by one single thread, the parallel embedding time is the sum of landmark embedding time and non-landmark embedding time on the slowest server. Table 3 shows that parallelizing embedding reduce the time taken from around 2 hours to less than 10 minutes for our largest graphs.

Next, we measure the average per-query response time for Leo, Orion, and the traditional Monte Carlo measurement method. Average response time is defined as the average time to compute the expected or stable random walk distance for a node pair. We average the computation time across 1000 random pairs of nodes. Table 3 shows the average response time using Leo, Orion, and the average response time to compute ground truth using the traditional method. As expected, response time on Leo is constant for different graph sizes, and is several orders of magnitude faster



(a) Egypt Graph (b) Planar Graph (c) Email Graph

Figure 13: Impact of embedding dimension on the accuracy of hitting time.

than traditional methods. For example, the time to estimate hitting time using Leo is 0.008ms ($8\mu s$), 8 orders of magnitude faster than the time required to compute the ground truth, $\sim$ 10 minutes. As expected, Table 3 also confirms that per-query response times on Leo and Orion are essentially identical. Once the upfront bootstrap phase is complete, Leo's response times of 8 microseconds for hitting time, PPR, and commute time queries make it more than capable of handling real-time queries on large graphs.

5.3 Applications

Now, we evaluate the utility of graph coordinate systems at the application level. We apply our embedding system into two popular applications, search ranking and link prediction. The two applications are built on expensive random walk distances, but are useful in practical search engines and social network analysis. Our experiments show that using graph coordinate systems in these applications can produce answers that closely approximate results derived from measured (ground-truth) random walk distances[4].

Search ranking. Ranking search results or entities based on their relevance is a fundamental problem in search engines [8] and recommendation systems [7]. For example, Google might return thousands of answers for the query "publications about social network published in 2012." For a better user experience, Google ranks the returned results based on the relevance metrics such that the most wanted results by the user should be prioritized. Among the metrics to quantify relevance, random walk distance is one of the most important and widely used metrics [8, 7, 6].

We implement a search ranking application to evaluate the impact of Leo. We choose a random node i to send out a query for which N answers are returned. Here, each answer is represented by a node in the graph. We then measure the random walk distances from node i to each node j in the set of N nodes. Finally, we rank the N nodes based their distances to node i and select the top K nodes as the best results for the query. We separately use commute time, hitting time and PPR as random walk distances in the ranking. When using commute or hitting time, we rank the result in an increasing order such that results with low commute time or hitting time are in top positions. In contrast, using PPR, we rank the results in descending order and the top K nodes have the highest PPR.

In our experiment, when a node sends out a query, $N = 2000$ random nodes return answers. We rank them using their distances to the query origin. Finally, we return the top $K = 50, 100, 1000$ answers, which is corresponding the top $5\% \sim 50\%$ answers. We repeat this experiment 2000 times. Each time we choose a random node to generate a query and rank the nodes using commute time, hitting time and PPR independently. For each random walk distance, we get two sets of top K nodes. One set is generated using measured actual distance and the other set is based on the distances estimated by Leo. We count the amount of overlap between the top K nodes in the two sets. We use the ratio of the number of overlapping nodes to the total number of top K nodes to quantify the accuracy. All our experiment is measured on the three real social graphs in Table 1.

We plot the results of hitting time, commute time and PPR in Figure 14. It shows that for all three distances, our system can more accurately approximate the ground truth as K is larger. For example, using commute time in Egypt graph, the accuracy increases from 70% to more than 80% when K increases from 500 to 1000.

Link prediction. Social network is a network with high dynamics. The addition of edges is one of the main reasons to cause the frequent changes in network. Predicting link creation in the future is one important problem for understanding network evolution and predicting network growth. [25] shows that node structure similarity in a network can be used to predict links. This paper uses several metrics to quantify nodes similarity. Among them, commute time, hitting time and PPR are three important metrics with high accuracy. However, the computation of these similarity metrics is very expensive. This motivates several studies [37, 39] to accelerate the random walk distance estimation and is one important application to evaluate the accuracy of the estimation. Thus, we use our system to estimate random walk distances to predict future links and compare the predicting accuracy to the accuracy generated by the actual distances. Again, we separately use hitting time, commute time and PPR in link prediction.

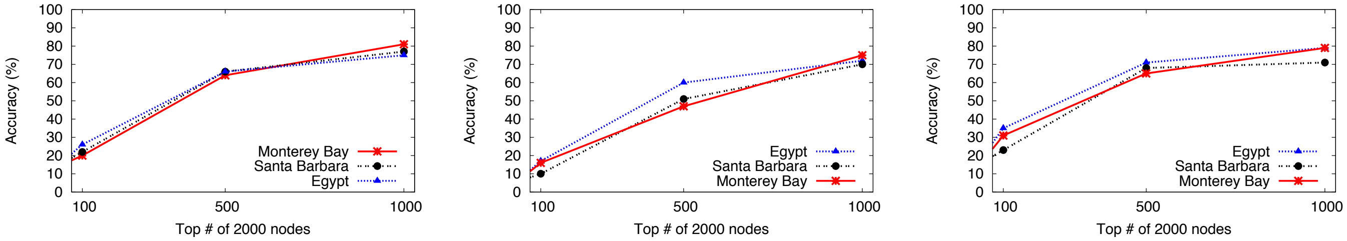
Our link prediction experiment is similar to [37]. We delete 10% random edges from each graph G for prediction, which results in a new graph G'. We test how accuracy of our system is in predicting the deleted 10% edges. Since measuring the actual distances for all nodes pairs as ground truth is costly in term of computation time, we only consider all pairs of nodes that have edges deleted as potential edges. Then we rank all potential edges based distances between their two endpoints in graph G'. Similar to search ranking, we rank commute time and hitting time in an increasing order while rank PPR in a descending order. We choose top M pairs of nodes in the ranked edge list, where M is the exact number of the 10% deleted edges in the graph. Again, we can have two sets of top M results using actual measured distances and estimated distances from Leo for each distance metric. For each set, we count how many of edges, *i.e.* node pairs, in this set overlap with the actual deleted edges and use the ratio of the overlapping edges to the total M edges as the accuracy metric. Thus, we can compare the prediction accuracy using Leo to the prediction accuracy using actual distances.

We run our experiment on the three real social graphs and show the results of the three random-walk distances in Table 4. For each metric, we find that the prediction accuracy of Leo is quite similar to the accuracy using actual distances. For example, in Egypt, using

[4] We use Leo with 10 dimensions.

Metric	Graphs	One-thread Bootstrap (hours)		Parallel Bootstrap (hours)		Per-query response (ms)		
		Precomputation	Embedding	Precomputation	Embedding	Ground Truth	Orion	Leo
Hitting Time	Egypt	168.12	1.59	1.88	0.12	566,359	0.0089	0.0089
	Amazon	157.23	1.93	1.67	0.13	162.08	0.0080	0.0085
	Web	235.12	2.11	2.39	0.15	1463.99	0.0084	0.0081
PPR	Egypt	11.20	1.63	0.16	0.15	17.5	0.0082	0.0087
	Amazon	12.09	1.94	0.17	0.14	18.2	0.0085	0.0088
	Web	12.15	2.12	0.17	0.15	17.9	0.0087	0.0085
Commute Time	Egypt	168.12	1.60	1.88	0.12	1,132,719	0.0082	0.0081
	Amazon	157.23	2.03	1.67	0.12	345.11	0.0082	0.0080
	Web	235.12	2.09	2.39	0.14	3,012	0.0083	0.0082

Table 3: Computation time of Leo on three largest real graphs, including bootstrap time and per-query response time.



(a) Hitting Time

(b) PPR

(c) Commute Time

Figure 14: Accuracy of Top k ranked node

Graphs	Hitting Time (%)		PPR (%)		Commute Time (%)	
	GT	Leo	GT	Leo	GT	Leo
MontereyBay	71.21	69.12	72.16	69.56	61.23	69.11
Santa Barbara	69.28	71.28	71.22	70.01	65.26	70.26
Egypt	73.52	69.93	71.56	72.38	68.01	67.98

Table 4: Link prediction application accuracy using hitting time, commute time, and PPR, based on Ground Truth (GT) and Leo.

hitting time in link prediction, the accuracy using actual distances is 73.52% while the accuracy of Leo is 69.93%. We notice that Leo outperforms actual distances in few cases. This is because the estimation error of Leo results in that some deleted edges are ranked higher than they should be. However, the number of this kind of node pairs is relatively small. The prediction accuracy of Leo is almost as accurate as the results using actual distances.

6. RELATED WORK

Applications Using Random Walks. Random walks appear in numerous applications in fields such as computer vision, data mining, network security and social network analysis. For example, work in computer vision [15] reliably extracts shape properties in Silhouettes using hitting time. [16] utilizes hitting time from all nodes to a chosen node as threshold to determine automated graph partitions. Commute time is an important way to track real-world multi-body motions [33] and image segmentations [32].

In data mining problems, standard clustering algorithms such as K-means produce more accurate results by replacing traditional distance with commute time [42]. Personal page rank has been used to improve partitioning in [2], and commute time has been used to detect global and local outliers in data [20]. Since random walks are resistant to noise and manipulation, they are also widely used to build robust reputation systems [17] and Sybil detection systems on social networks [43, 44]. In social networks, commute time, hitting time and PPR are important metrics to accurately perform link prediction over time [25].

Random Walk Distance Estimation. Given the prevalence of Personal PageRank in search engines and recommendation systems, researchers have developed two general approaches to compute PPR, *i.e.* linear algebraic optimization [19, 29] and Monte Carlo approximation algorithms [13, 3]. Monte Carlo algorithms to compute PPR can be significantly sped up using a variety of techniques, ranging from parallelization via MapReduce [4] to improved bounds for distributed algorithms [11].

Since commute time is symmetric, many have tried to approximate it using fast matrix computation. Standard matrix computations based on matrix inverse require $O(n^3)$ time, which does not scale. One solution is to produce fast approximations using the Lanczos process [6]. Since network effective resistance is analogous to commute time, [40] uses graph sparsification to compute effective resistances between any pair of nodes in $O(\log n)$ time. Finally, [37] focuses on efficiently computing hitting time and commute time within a fixed number of T hops, instead of a generalized query between any two nodes.

Distance Embedding. Distance embedding methods have been used to approximate delays between machines on a network, in the form of network coordinate systems [31, 28, 10, 27]. These systems embed Internet latency, *i.e.* round-trip time (RTT), into geometric spaces. To account for violations of the triangle inequality, [10] uses a height value to model congestion at the network edge.

Some initial work has applied embedding methods to shortest path queries, with significant limitations in scalability. [34] embeds metrics in small graphs into a Euclidean space, and [35] uses a network structure index (NSI) to compute node position. [9] proposes a greedy method to capture dynamic graphs in a hyperbolic space. However, all three methods cannot scale to medium or large graphs in today's social networks. Orion [47] and Rigel [48] are Graph Coordinate Systems that focus on approximating shortest paths using Euclidean and Hyperbolic spaces.

7. CONCLUSION

Random walk distances are important for their ability to capture not only distance but also affinity. Unfortunately, they are prohibitively expensive to compute even moderately sized graphs. Our work explores novel techniques to accurately approximate three random walk distances: hitting time, commute time and personalized PageRank.

Our work shows that naive graph embeddings fail to capture the asymmetry of these distances. Our insight is to identify, per-direction, per-node random walk costs that can be accurately captured and modeled. We model them as two additive "height vectors" that are added to random walk distance estimates. In an abstract sense, one captures the cost of leaving the subgraph around the source node, and one captures the cost of finding the destination node in its local subgraph. We show that these factors change dramatically based on the type of graph, and that for small-world graphs, asymmetric hitting time is dominated by a single per-destination cost. We propose Leo, a new embedding system leveraging these insights to accurately predict random walk distances. Our evaluation shows that after the upfront embedding cost, Leo provides highly accurate predictions in a few microseconds, 8 orders of magnitude faster than existing methods for computing ground truth. Source code at `http://sandlab.cs.ucsb.edu/rigel`.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback. This work is supported in part by DARPA GRAPHS (BAA-12-01) and NSF grants CNS-1224100 and IIS-0916307. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

8. REFERENCES

[1] R. Albert, H. Jeong, and A. Barabasi. Diameter of the world-wide web. *Nature*, pages 130–131, 1999.

[2] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *Proc. of FOCS*, pages 475–486, 2006.

[3] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. Technical report, SIAM J. Numer. Anal, 2005.

[4] B. Bahmani, K. Chakrabarti, and D. Xin. Fast personalized pagerank on mapreduce. In *Proc. of SIGMOD*, pages 973–984, 2011.

[5] Z. Bar-Yossef and L.-T. Mashiach. Local approximation of pagerank and reverse pagerank. In *Proc. of CIKM*, pages 279–288, 2008.

[6] F. Bonchi et al. Fast matrix computations for pairwise and columnwise commute times and katz scores. *Internet Mathematics*, 8(1-2):73–112, 2012.

[7] M. Brand. A random walks perspective on maximizing satisfaction and profit. In *Proc. of SDM*, 2005.

[8] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proc. of WWW*, pages 571–580, 2007.

[9] A. Cvetkovski and M. Crovella. Hyperbolic embedding and routing for dynamic graphs. In *Proc. of INFOCOM*, pages 1647–1655, 2009.

[10] F. Dabek et al. Vivaldi: A decentralized network coordinate system. In *Proc. of SIGCOMM*, pages 15–26, 2004.

[11] A. Das Sarma, A. R. Molla, G. Pandurangan, and E. Upfal. Fast distributed pagerank computation. In *ICDCN*, pages 11–26, 2013.

[12] S. N. Dorogovtsev and J. F. F. Mendes. Evolution of networks. In *Proc. of Adv. Phys*, pages 1079–1187, 2002.

[13] D. Fogaras et al. Towards scaling fully personalized pageRank: algorithms, lower bounds, and experiments. *Internet Math.*, 2(3):333–358, 2005.

[14] S. Gaito et al. On the bursty evolution of online social networks. In *Proc. of HotSocial*, pages 1–8, 2012.

[15] L. Gorelick et al. Shape representation and classification using the poisson equation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(12):1991–2005, 2006.

[16] L. Grady and E. Schwartz. Isoperimetric partitioning: a new algorithm for graph partitioning. *SIAM Journal of Scientific Computing*, 27(6):1844–1866, 2006.

[17] J. Hopcroft and D. Sheldon. Manipulation-resistant reputations using hitting time. *Internet Math*, 5(1-2):71–90, 2008.

[18] G. Jeh and J. Widom. Scaling personalized web search. In *Proc. of WWW*, pages 271–279, 2003.

[19] S. Kamvar et al. Extrapolation methods for accelerating pagerank computations. In *Proc. of WWW*, pages 261–270, 2003.

[20] N. L. D. Khoa and S. Chawla. Robust outlier detection using commute time and eigenspace embedding. In *Proc. of PAKDD*, pages 422–434, 2010.

[21] P. Lawrence et al. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.

[22] J. Leskovec, L. Adamic, and B. Adamic. The dynamics of viral marketing. *ACM TWEB*, (1), 2007.

[23] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proc. of KDD*, pages 177–187, 2005.

[24] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1), 2007.

[25] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proc. of CIKM*, pages 556–559, 2003.

[26] L. Lovász. Random walks on graphs: A survey. *Bolyai Society Mathematical Studies*, 2:1–46, 1993.

[27] C. Lumezanu et al. Measurement manipulation and space selection in network coordinates. In *Proc. of ICDCS*, pages 361–368, 2008.

[28] M.Costa et al. Pic: Practical internet coordinates for distance estimation. In *Proc. of ICDCS*, pages 178–187, 2004.

[29] F. McSherry. A uniform approach to accelerated pagerank computation. In *Proc. of WWW*, pages 575–582, 2005.

[30] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, Jan. 1965.

[31] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *Proc. of INFOCOM*, pages 170–179, 2002.

[32] H. Qiu and E. Hancock. Image segmentation using commute times. In *Proc. of BMVC*, pages 929–938, 2005.

[33] H. Qiu and E. Hancock. Robust multi-body motion tracking using commute time clustering. *ECCV*, pages 160–173, 2006.

[34] S. Rao. Small distortion and volume preserving embeddings for planar and euclidean metrics. In *Proc. of SCG*, pages 300–306, 1999.

[35] M. J. Rattigan et al. Using of structure indices for efficinet approximation of network properties. In *Proc. of KDD*, pages 357–366, 2006.

[36] M. Saerens et al. The principal components analysis of a graph, and its relationships to spectral clustering. In *Proc. of ECML*, pages 371–383, 2004.

[37] P. Sarkar and . W. Moore. A tractable approach to finding closest truncated-commute-time neighbors in large graphs. In *Proc. of UAI*, pages 335–343, 2007.

[38] T. Sarlós et al. To randomize or not to randomize: space optimal summaries for hyperlink analysis. In *WWW*, pages 297–306, 2006.

[39] H. Song et al. Scalable proximity estimation and link prediction in online social networks. In *Proc. of IMC*, pages 322–335, 2009.

[40] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. In *Proc. of STOC*, pages 563–568, 2008.

[41] C. Wilson, B. Boe, A. Sala, K. P. N. Puttaswamy, and B. Y. Zhao. User interactions in social networks and their implications. In *Proc. of EuroSys*, pages 205–218, April 2009.

[42] L. Yen et al. Clustering using a random walk based distance measure. In *Proc. of ESANN*, pages 317–324, 2005.

[43] H. Yu et al. Sybilguard: defending against sybil attacks via social networks. In *Proc. of SIGCOMM*, pages 267–278, 2006.

[44] H. Yu et al. Sybillimit: A near-optimal social network defense against sybil attacks. In *Proc. of IEEE S&P*, pages 3 – 17, 2008.

[45] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. of infocom*, pages 594–602, 1996.

[46] X. Zhao, A. Sala, C. Wilson, X. Wang, S. Gaito, H. Zheng, and B. Y. Zhao. Multi-scale dynamics in a massive online social network. In *Proc. of IMC*, pages 171–184, 2012.

[47] X. Zhao, A. Sala, C. Wilson, H. Zheng, and B. Y. Zhao. Orion: Shortest path estimation for large social graphs. In *WOSN*, 2010.

[48] X. Zhao, A. Sala, H. Zheng, and B. Y. Zhao. Efficient shortest paths on massive social graphs. In *CollaborateCom*, pages 77–86, 2011.