A Sublinear Algorithm for Approximate Shortest Paths in Large Networks

A Sublinear Algorithm for Approximate Shortest Paths in Large Networks

Sabyasachi Basu*
UC Santa Cruz
Santa Cruz, CA, USA
sbasu3@ucsc.edu

Nadia Kōshima*
Massachusetts Institute of Technology
Cambridge, MA, USA
nadianw36@gmail.com

Talya Eden
Bar-Ilan University
Ramat Gan, Israel
talyaa01@gmail.com

Omri Ben-Eliezer
Technion – Israel Institute of Technology
Haifa, Israel
omribene@cs.technion.ac.il

C. Seshadhri
UC Santa Cruz
Santa Cruz, CA, USA
Amazon
Cupertino, CA, USA
sesh@ucsc.edu

Abstract

Computing distances and finding shortest paths in massive real-world networks is a fundamental algorithmic task in network analysis. There are two main approaches to solving this task. On one end are traversal-based algorithms like bidirectional breadth-first search (BiBFS), which have no preprocessing step but are slow on individual distance inquiries. On the other end are indexing-based approaches, which create and maintain a large index. This allows for answering individual inquiries very fast; however, index creation is prohibitively expensive. We seek to bridge these two extremes: quickly answer distance inquiries without the need for costly preprocessing.

We propose a new algorithm and data structure, `WormHole`, for approximate shortest path computations. `WormHole` leverages structural properties of social networks to build a sublinearly sized index, drawing upon the core-periphery decomposition of Ben-Eliezer et al [10]. Empirically, `WormHole`'s preprocessing time improves upon index-based solutions by orders of magnitude: indexing billion edges graphs takes only a few minutes. Real time performance is consistently much faster than in BiBFS. The acceleration comes at the cost of a minor accuracy trade-off. We complement these empirical results with provable theoretical guarantees.

CCS Concepts

• **Information systems** → **Web mining**; • **Information system** → **Data mining**.

Keywords

Graph algorithms, Data mining, Social networks

ACM Reference Format:
Sabyasachi Basu*, Nadia Kōshima*, Talya Eden, Omri Ben-Eliezer, and C. Seshadhri. 2025. A Sublinear Algorithm for Approximate Shortest Paths in Large Networks. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining (WSDM '25), March 10–14, 2025, Hannover, Germany*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3701551.3703512

WSDM '25, March 10–14, 2025, Hannover, Germany

ACM ISBN 979-8-4007-1329-3/25/03
https://doi.org/10.1145/3701551.3703512

1 Introduction

Scalable computation of distances and shortest paths in a large network is one of the most fundamental algorithmic challenges in graph mining and graph learning tasks, with applications across science and engineering. Examples of such applications include the identification of important genes or species in biological and ecological networks [18], driving directions in road networks [1–3], redistribution of task processing from mobile devices to cloud [58], computer network design and security [24, 30, 31, 57], and identifying a set of users with the maximum influence in a social network [32, 56], among many others. Thus, a long line of work [4, 21, 25, 28, 61] has developed over the years, constructing scalable algorithms for distance computation for a variety of real-life tasks.

The simplest methods for answering a shortest path inquiry (s, t) are based on *traversals*, with breadth first search (BFS) as the most basic example. A popular modification, Bidirectional BFS (`BiBFS`), runs BFS from both s and t, alternating between the two until both ends meet. It has well been observed in the literature that `BiBFS` performs surprisingly well for shortest path inquiries on a wide range of networks (see, e.g., [8, 12, 61] and the many references within). Still, pure traversal-based approaches like `BiBFS` are too slow for practical purposes that require answering a large number of shortest pair inquiries.[1]

Index-based approaches (e.g., [4, 37, 61]) sit at the other end of the spectrum. The basic idea is to preprocess the network (before any inquiries arrive) to create an *index*. The index, in turn, supports extremely fast real-time computation of distances. This line of work has been investigated extensively in recent years, with Pruned Landmark Labeling (`PLL`, Akiba et al. [4]) being perhaps the most influential approach.

[1]To avoid confusion, throughout this paper we use the term *inquiry* to indicate a request (arriving as an input in real-time to our data structure) to compute a short path $\mathsf{SP}(s, t)$ between s and t. The term *query* refers to the act, taken by the algorithm itself, of retrieving information about a specific node. For more details on the query model we consider, see §1.2.)

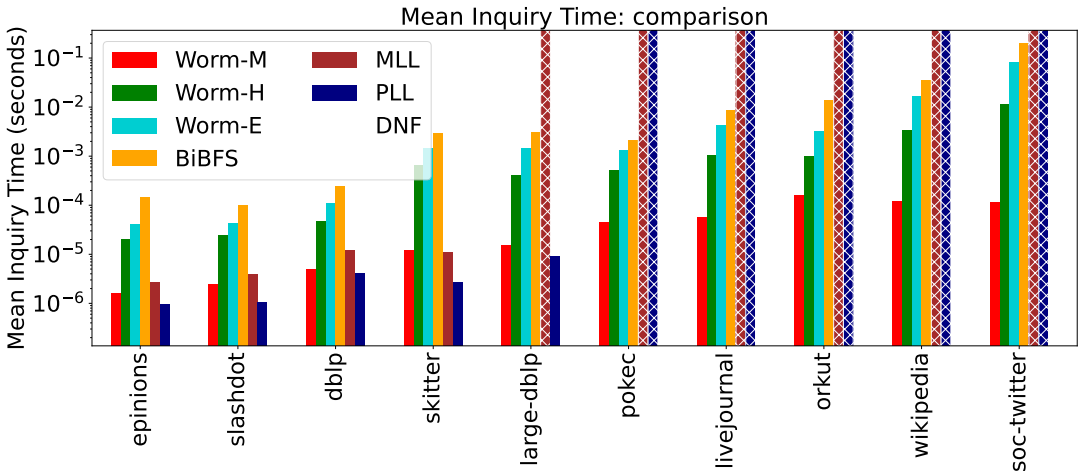


Figure 1: Average running time per shortest path inquiry for three variants of `WormHole`, as compared to index-based (MLL [61] and PLL [4]), and traversal-based (`BiBFS`) methods. `PLL` only finds distances, not paths. DNF marks that the preprocessing (index construction) step did not finish. All three of our variants outperformed `BiBFS` consistently. Index-based methods have inquiry time comparable to $\texttt{WormHole}_M$ on smaller networks, but did not finish on larger networks.

Although index-based approaches have been useful in practice due to their extremely fast inquiry time, these approaches have a number of significant shortcomings. The most central weakness is a very high space (memory) cost. Typically, an index-based method selects a subset of nodes, called *landmarks*; computes all shortest paths among them; and keeps an index of the distance of every node in the network to every landmark. Naively, the space (memory) requirement can be as bad as *quadratic* in n. Despite several improvements to beat the quadratic footprint, existing hub and landmark-based approaches methods continue to have high cost, and can become infeasible even for moderately-sized graphs [40].

Additionally, most index-based approaches only return distances in the graph, and not the paths themselves. Zhang et al. [61] showed how index-based solutions can be adapted to also output shortest paths, albeit at a very high space cost. They proposed a new approach called Monotonic Landmark Labelling (`MLL`) for saving on the index construction space cost. While `MLL` is the current state of the art for computing shortest path queries at scale, it is again index-based, meaning that the preprocessing cost is still high. Beyond the computational constraints, index-based approaches assume access to the whole network during preprocessing, an assumption that is simply unrealistic in numerous applications [10, 14, 26, 27, 59].

The limitations of indexing-based and traversal-based methods give rise to a natural question of whether there is a middle ground solution, with preprocessing that is more efficient than in index-based approaches and inquiry time that is faster than in traversal-based approaches. Namely, we ask:

Can we answer shortest-path inquiries very quickly, without constructing an expensive index, or even seeing the whole graph?

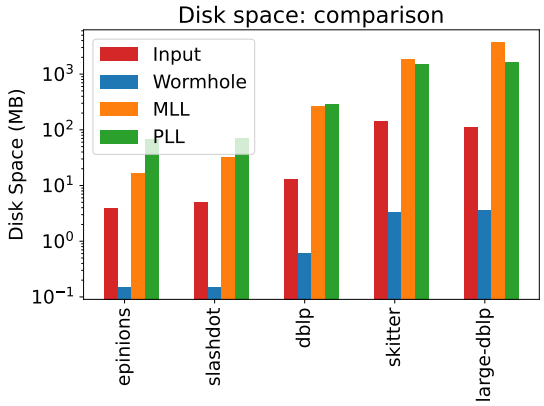
A general solution that works for all graphs seems impossible; however, real-world social and information networks admit structure that can be exploited. In this work, we address this question positively for a slightly relaxed version of the shortest path problem on such networks. Inspired by the *core-periphery structure* of large networks [43, 50, 52, 62], we provide a solution which constructs a *sublinearly-sized index* that does not need to access the whole network during its construction. The algorithm returns *approximate* shortest paths, where the approximation error is additive and very small (almost always zero or one).

1.1 Our Contribution

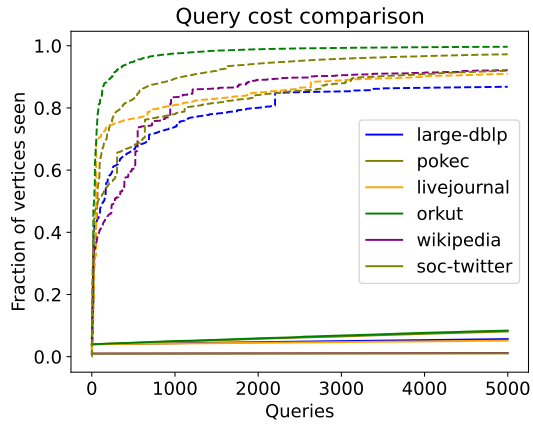
We design a new algorithm, `WormHole`, that creates a data structure allowing us to answer multiple shortest path inquiries by exploiting the typical structure of many social and information networks. The algorithm utilizes these insights to cleverly decompose the graph and calculate approximate shortest paths. We discuss the algorithm and various steps in detail in §3.

`WormHole` is simple, easy to implement, and theoretically backed. We provide several variants of it, each suitable for a different setting, and conduct extensive empirical evaluation. Below are some of the key features of `WormHole`:

- **Performance-accuracy tradeoff.** To the best of our knowledge, `WormHole` is the first approximate *sublinear* shortest paths algorithm in large networks. We have a trade-off between preprocessing time/space and per-inquiry time, which we explore in detail. Our most accurate (but slowest) variant, $\texttt{WormHole}_E$, has near-perfect accuracy: more than 90% of the inquiries are answered *with no additive error*, and in *all* networks, more than 99% of the inquiries are answered with additive error at most 2. See Table 1 for more details.
- **Extremely rapid setup time.** Our longest index construction time was just two minutes even for billion-edged graphs. `PLL` and `MLL` timed out on half of the networks that we tested. In moderately sized graphs where `PLL` and `MLL` did finish their runs, `WormHole` index construction was ×100 faster. This rapid setup time is achieved due to the use of a sublinearly-sized index, which may be as small as 1% of the vertices.



(a) Stored index size on disk for different algorithms.



(b) Fraction of vertices seen by WormHole. The dotted lines represent BiBFS and solid lines are WormHole.

Figure 2: (a) Disk space use for different methods. The indexing based methods did not terminate on graphs larger than these. For WormHole, we consider the sum of C_{in} and C_{out} binary files. PLL solves the easier problem of computing the distances. The red bar "Input" is the size of the edgelist. (b) The number of unique vertices queried by BiBFS (dotted lines) and WormHole (solid lines) . BiBFS ends up seeing between 70% and 100% of the vertices in just a few hundred inquiries; we are well below 20% even after 5000 inquiries.

- **Fast inquiry time.** Compared to BiBFS, the vanilla version $\texttt{WormHole}_E$ (without any index-based optimizations) is ×2 faster for almost all graphs and more than ×4 faster on the three largest graphs that we tested. A simple variant $\texttt{WormHole}_H$ achieves an order of magnitude improvement at some cost to accuracy: consistently 20× faster across almost all graphs, and more than 180× for the largest graph we have. See Table 1 for a full comparison.
- **Combining WormHole and the state of the art.** We also implement a variant of WormHole that runs MLL on the core (only). We call this variant $\texttt{WormHole}_M$. This variant achieves a "best of all worlds" performance in terms of running times. The inquiry times of $\texttt{WormHole}_M$ are comparable to MLL (with the same accuracy guarantee as $\texttt{WormHole}_H$), while its setup cost is orders of magnitude smaller. In particular, $\texttt{WormHole}_M$ runs on massive graphs in which MLL does not terminate. We explore this combined approach in §5.3.
- **Sublinear query complexity.** The query complexity refers to the number of vertices visited by the algorithm. In a query access model where querying a node reveals its list of neighbors (see §1.2), the query complexity of our algorithm scales very well with the number of distance/shortest path inquiries. To answer 5000 approximate shortest path inquiries, our algorithm observes between 1% and 20% of the nodes for most networks. In comparison, BiBFS sees more than 90% of the graph to answer a few hundred inquiries. See Figure 2 and Figure 4 for a comparison.
- **Provable guarantees on error and performance.** In §4 we prove a suite of theoretical results complementing and explaining the empirical performance. We show that for the Chung-Lu model of random graphs with a power-law degree distribution [15–17], WormHole incurs very low error ($O(\log \log n)$) and is provably sublinear in query complexity ($n^{o(1)}$). Moreover, any method that avoids preprocessing *must* have $n^{\Omega(1)}$ query cost per inquiry.

1.2 Setting

We consider the problem of constructing a data structure for *approximately* answering shortest-path inquiries between vertex pairs (s, t) in an undirected graph G, given query access to G.

Query model: Access to the network is given through the standard node query model [10, 14], where we start with an arbitrary seed vertex as the "access point" to the network, and querying a node v reveals its list of neighbors $\Gamma(v)$. Unlike existing index-based solutions, which perform preprocessing on the whole graph, we aim for a solution that queries and stores only a small fraction of the nodes in the network.

Objective: Following the initialization of the data structure, the task is to answer multiple shortest path inquiries, where each inquiry $\texttt{SP}(s, t)$ needs to be answered with a valid path $s = p_0 p_1 \ldots, p_\ell = t$, and the objective is to minimize the mean additive error measured over all inquiries. The additive error for an inquiry $\texttt{SP}(s, t)$ is the difference between the length of the returned s–t path and the actual distance. Depending on the specific application, one would like to minimize (a subset of) the additive error, running time, memory and/or node queries.

Core-periphery structure: The degree distribution in social and information networks often follows a power-law distribution with exponent $2 < \beta < 3$, which results in a *core-periphery* structure [9, 43, 50, 52, 62]. The core is a highly connected component with high degree nodes and good expansion properties, while the periphery is a collection of small, poorly connected components of low degree.

2 Related Work

There is a vast body of research on shortest paths and distance computations. Here we review some of the most closely related works to ours. For a more comprehensive overview, see the surveys [42, 55] and references therein.

Index-based approaches: As mentioned earlier, a ubiquitous set of algorithms are based on landmark/sketch approaches [38, 60, 61], with Pruned Landmark Labeling (PLL) [4] being perhaps the

most influential one. These algorithms follow a two-step procedure: the first step generates an ordering of the vertices according to importance (based on different heuristics), and the the second step generates labeling from pruned shortest path trees constructed according to the ordering. Then the shortest distance between an arbitrary pair of vertices s and t can be answered quickly based on their labels. However, even with pruning, PLL requires significant setup time. Hence, there have been many attempts to parallelize it [29, 38, 39].

Embedding based approaches: Recent approaches leverage embeddings of graphs to estimate shortest paths. Like in representation learning, they seek to find efficient representations of distances between pairs of nodes [11, 33, 63, 64] and explore extensions utilizing deep learning [35, 47, 48].

BFS-based approaches: Another set of algorithms is based on BFS or BiBFS, as they are exact methods with no preprocessing step. There has been substantial work in the last few years proving that BiBFS has sublinear $n^{1-\Omega(1)}$) time with high probability for several graph families [8, 11, 12]. The work of [8] also proves query lower bounds of the form $n^{\Omega(1)}$ for traversal-based algorithms in Erdős-Renyi and d-regular random graphs.

Routing in road networks: Finding the shortest path is a fundamental problem for routing tasks on maps. There is a rich family of work using approaches based on the highway dimension [1–3] for routing in road networks. We emphasize that the road networks are nearly planar, and structural characteristics of these network types differ considerably from our setting.

Core-periphery based approaches: Allavena et al. [7] had similar observations about a small core in random power law graphs. Several other works exploit the core-periphery structure of networks [5, 13, 37]. Brady and Cohen [13] use compact routing schemes to design an algorithm with small additive error based on the resemblance of the periphery to a forest. Akiba, Sommer and Kawarabayashi [5] exploit the property of low tree-width outside the core, and in [37], the authors design a Core-Tree based index in order to improve preprocessing times on massive graphs. We note that in all these results, the memory overhead is super-linear.

Worst case graphs: On the theoretical side, there have been many results on exact and approximate shortest paths in worst-case graphs, e.g., [19, 20, 22, 49, 51, 66]. We point the readers to Zwick's survey [65] on exact and approximate shortest-path algorithms, and Sen's survey [53] on distance oracles for general graphs with an emphasis on lowering pre-processing cost. Notably, because we make a structural assumption that is common in large networks, namely, a core-periphery structure, both our results and algorithm substantially differ from the worst-case theoretical literature.

3 Algorithm

WormHole builds a hierarchical core-periphery structure with a sublinear inner ring. It uses this structure to answer shortest path inquiries. There are two phases: (i) A preprocessing step where we decompose the graph into three partitions, storing only the smallest one: an inner core which consists of a highly dense subgraph *of sublinear size*; and (ii) answering inquiries: here the algorithm (approximately) answers shortest path inquiries of the form $\mathsf{SP}(s,t)$ for arbitrary vertex pairs (s,t).

The Structural Decomposition Phase. To capture the inner-core in sublinear time, we follow the structural decomposition provided in [10] and denote it as the CoreGen procedure. This decomposition stratifies our graph into three sections: the inner ring, C_{in}, a dense component with the highest degree nodes, the outer ring ,C_{out}, the set of neighbors of the inner ring, and the rest of the vertices which form the periphery, $\mathcal{P}$. The procedure iteratively expands the inner ring, starting from an arbitrary seed vertex, given as input v. The second parameter, p, controls the size of inner ring, in terms of a percentage of the vertex set, and the process is iterated till that size is reached. The psuedocode is given in Algorithm 2 in the supplement.

We emphasize that *this step is done only once*; all subsequent operations are done using this precomputed decomposition.

Algorithm 1 $\mathtt{WormHole}(s,t,C_{\mathsf{in}},C_{\mathsf{out}},G)$: final algorithm

$T(s) = \Gamma(s), T(t) = \Gamma(t)$
$C(s), C(t) \leftarrow \emptyset$
while $T(s) \cap T(t) = \emptyset$ or if both $C(s), C(t) = \emptyset$ **do**
  for both s and t (denoted by i in the next block) **do**
    if $C(i) = \emptyset$ and $\Gamma(T(i)) \cap C_{\mathsf{in}} \neq \emptyset$ **then**
      Expand $T(i)$ by a single level
      $C(i) \leftarrow \Gamma(T(i)) \cap C_{\mathsf{in}}$
      stop further expansion of $T(i)$
    else if $C(i) = \emptyset$ **then**
      Expand $T(i)$ by a single level
    end if
  end for
end while
if $T(s) \cap T(t) \neq \emptyset$ **then**
  return $\mathsf{SP}(s,t)$ through $T(s) \cap T(t)$
else
  Expand the BiBFS tree from $C(s), C(t)$ in $G[C_{\mathsf{in}}]$
  return $\mathsf{SP}(s,t)$ through $T(s) \cup G[C_{\mathsf{in}}] \cup T(t)$
end if

The Routing Phase. The approach is simple: assume the preprocessing phase acquires the inner ring. Upon an inquiry (s,t), the algorithm starts two BFS trees from both s and t. It then expands the tree at each step till one of the followings happens:

- the two trees intersect, or
- the trees reach the outer ring.

If the search trees in the former do not intersect, WormHole mandatorily routes shortest paths through the inner ring. Once in the inner ring, it computes the exact shortest path through it.

Variants of WormHole. In the default variant of Algorithm 1, $\mathtt{WormHole}_E$, we use BiBFS as a primitive to compute the shortest path between two vertices in C_{in}; see [46] for a full description of BiBFS. The theoretical analysis in §4 considers this variant.

$\mathtt{WormHole}_H$ is a simple variant where we start the expansion of the BiBFS trees in $G[C_{\mathsf{in}}]$ only from the vertex $v_s = \mathrm{argmax}_{v \in C(s)} deg(v)$ (and similarly v_t for $C(t)$, the highest degree vertices). We note that for this variant, the theoretical accuracy bound in §4 no longer holds. An empirical comparison of the two is done in §5.1 and Table 1.

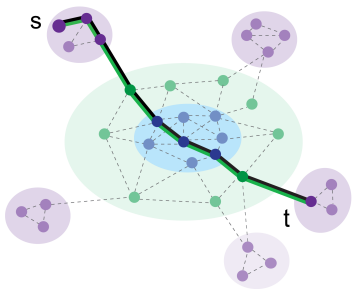


Figure 3: An example of our algorithm's traversal from source s to target t in two separate peripheral components. The colored blue, green, and purple regions respectively represent C_{in}, C_{out}, and $\mathcal{P}$. If the true shortest path (in green) passes through the core, our algorithm finds it (in black). Other cases are illustrated in the supplement.

In WormHole$_M$, we combine WormHole with an index-based algorithm restricted to the core. While indexing-based algorithms are quite expensive, in terms of both preprocessing and space cost, they lead to much lower times per inquiry. Therefore this variant is suitable when faster inquiry times are preferred. WormHole$_M$ makes the index creation cost substantially lower (compared to generating it for the entire graph) while providing massive speedups for answering shortest path inquiries. We discuss this briefly in §5.3.

4 Theoretical Analysis

It is a relatively standard observation that many social networks exhibit, at least approximately, a power-law degree distribution (see, e.g., [9] and the many references within). The Chung-Lu model [41] is a well studied random graph model with this property. We provide a proof-of-concept for the correctness and fast performance of WormHole on graphs generated according to this model.

Preliminaries. A network is said to have a power-law degree distribution when for sufficiently large k, the fraction of nodes with degree k, denoted $p(k)$, follows a power-law. That is, $p(k) \propto k^{-\beta}$. Multiple studies have shown that real world graphs indeed follow (or approximately follow) a power-law degree distribution, and the exponent β is typically in the range $2 < \beta < 5$ or so [6, 23, 44, 54]. Theoretically, the regime $2 < \beta < 3$ is the most interesting, and thus we focus our study on this regime.

In a series of works, Chung and Lu suggested a model to generate graphs according to a power-law degree distribution, in order to generate massive graphs that capture properties of real-world graphs [15–17]. In this model, one is given a list of n expected degrees, $w_1 \geq w_2 \geq \ldots \geq w_n$, where in our case we assume that the expected degrees follow a power-law degree distribution with $2 < \beta < 3$. Then, for each pair of nodes i, j, the edge between them is instantiated with probability $\frac{w_i \cdot w_j}{W}$, where $W = \sum_{i=1}^{n} w_i$ (and it is assumed that for any i, j, $w_i \cdot w_k \leq W$). Note that indeed by this definition, it holds that for every vertex v, $\mathrm{E}[d(v)] = w(v)$, and therefore that $\mathrm{E}[|E(G)|] = W/2$. We denote this distribution on graphs as $\mathcal{G}^{\beta}_{CL}$, and denote a graph drawn from it by $G \sim \mathcal{G}^{\beta}_{CL}$.

In all that follows we assume that $G \sim \mathcal{G}^{\beta}_{CL}$, and the probabilities are taken over the generation process. We let $\gamma = 1/\log\log n$, and let C_{CL} denote the set of vertices with degree at least $t = n^{\gamma}$.

Theorem 4.1 (Theorem 4 in [16]). *Suppose $G \in \mathcal{G}^{\beta}_{CL}$ for $2 < \beta < 3$, with average degree d strictly greater than 1, and maximum degree $d_{max} > \log n/\log\log n$. Then almost surely the diameter is $\Theta(\log n)$, the diameter of the C_{CL} core is $O(\log\log n)$ and almost all vertices are within distance $O(\log\log n)$ to C_{CL}.*

Claim 4.2 (Fact 2 in [15]). *With probability at least $1 - 1/n$, for all vertices $v \in V$ such that $w(v) \geq 10\log n$, $d(u) \in \left[\frac{1}{2}w(u), 2w(u)\right], |E(G)| \in [W/4, W]$, and for all vertices with $w(v) \leq 10\log n$, $d(v) \leq 20\log n$.*

Sublinearity of Inner Ring. Chung and Lu proved [15–17] that for graphs $G \sim \mathcal{G}_{CL}$ and $2 < \beta < 3$, there exists a core with small diameter, so that most of the vertices in the graph are close to it. We extend their result to show that this core is of sublinear size and that CoreGen indeed captures it; i.e., $C_{\text{CL}} \subseteq C_{\text{in}}$.

Theorem 4.3 (WormHole inner ring $\supseteq$ Chung-Lu core). *Consider a graph G generated according to the Chung-Lu process with $\beta > 2$, and suppose we run WormHole on G with an inner ring of size $n^{1-\Theta(1/\log\log n)}$. Then with high constant probability, C_{in} contains all vertices with degree greater than $n^{1/\log\log n}$.*

Proof Sketch. Consider a process P which simultaneously constructs G as the core grows as outlined in CoreGen: Initially the graph G has no edges, and C_{in} is an empty set. Every time a vertex v is added to C_{in}, P iterates over all nodes $u \in V \setminus C_{\text{in}}$ and draws a coin with probability $\frac{w(v)\cdot w(u)}{2W}$ to determine whether the edge (u, v) is in E or not. Once C_{in} is acquired (i.e., when C_{in} is of size $t = 100\log n \cdot n^{1-1/\log\log n}$), P iterates over all pairs of vertices that were not yet examined and determines their edges. Clearly this process generates a graph according to the distribution $G \sim \mathcal{G}^{\beta}_{CL}$ (since we only changed the order in which the edges were decided).

Let d denote the expected average degree in the graph, and let C^{ℓ}_{in} denote the set of vertices added to C_{in} after ℓ steps of the above process. Finally, let $\gamma = 1/\log\log n$. It is easy to show that with high probability, except for a negligible fraction, all vertices added to the core have expected degree $w(V) \geq d$. Conditioned on this event, for every u with $w(u) \geq n^{\gamma}$, every time a vertex v is added to the core, v creates an edge with u with probability

$$\frac{d(u)\cdot d(v)}{W} \geq \frac{n^{\gamma}\cdot d}{nd} = \frac{1}{n^{1-\gamma}}.$$

Let χ_i denote the event that the i-th vertex added to C_{in} creates an edge with u. Then by the above, $\mathrm{E}[\chi_i] \geq \frac{1}{n^{1-\gamma}}$. Observe that $d_{C_{\text{in}}}(u) = \sum_i \chi_i = \frac{|C^{\ell}_{\text{in}}|}{n^{1-\gamma}}$, and set $\mu := \frac{|C^{\ell}_{\text{in}}|}{n^{1-\gamma}}$. Note that the χ_i variables are independent $\{0, 1\}$ random variables, and that for $\ell = c\log n \cdot n^{1-\gamma}$, $\mu \geq c\log n$. Therefore, by Chernoff,

$$\Pr\left[\left|d_{C^{\ell}_{\text{in}}}(u) - \mu\right| > \frac{1}{4}\mu\right] < 2\exp\left(\frac{-\frac{1}{16}\cdot\mu}{3}\right) \leq \frac{1}{n^2}.$$

That is, we get that with probability $1 - 1/n$, all vertices with expected degree $w(u) \geq n^\gamma$ have $d_{C_{\text{in}}}(u) \geq \frac{3}{4}n^\gamma$. A similar analysis can be used to prove that all vertices u with expected degree $w(u) < n^\gamma/8$ will have $d_{C_{\text{in}}^{2\ell}}(u) \leq \frac{1}{2}n^\gamma$ with high probability. Hence, after performing additional ℓ steps, we will have that with high probability, all vertices with expected degree at least n^γ will be added to $C_{\text{in}}^{2\ell}$ before any vertex with expected degree less than $n^\gamma/8$. Finally, using Lemma 4.2, the degrees of all vertices will either be as expected up to a constant multiplicative factor, or have an expected degree below, say $n^\gamma/8$, and actual degree not higher than $n^\gamma/2$. □

Approximation Error. Now that we have a sublinear inner ring that contains the Chung-Lu core, it remains to show that routing paths through it incurs only a small penalty. Intuitively, the larger the inner ring, the easier this is to satisfy: if the inner ring is the whole graph, the statement holds trivially. Therefore the challenge lies in showing that we can achieve a strong guarantee in terms of accuracy even with a sublinear inner ring. We prove in Section 6 of the supplement that WormHole incurs an additive error at most $O(\log\log n)$ for all pairs, which is much smaller than the diameter $\Theta(\log n)$.

Theorem 4.4 (Good additive error). *If $C_{CL} \subseteq C_{in}$, then with high probability, for all pairs (s,t) of nodes, the additive error of our algorithm for the inquiry $\mathsf{SP}(s,t)$ is at most $O(\log\log n)$.*

The proof of Theorem 4.4, together with the proofs of Theorems 4.5 and 4.6 below, appear in the supplementary material.

The above result holds with high probability even *in the worst case*. Namely, for *all* pairs (s,t) of vertices in the graph, the length of the path returned by WormHole is at most $O(\log\log n)$ higher than the actual distance between s and t. This trivially implies that the average additive error of WormHole is, with high probability, bounded by the same amount.

Query Complexity. Recall the node query model in this paper (see §1.2): starting from a single node, we are allowed to iteratively make queries, where each query retrieves the neighbor list of a node v of our choice. We are interested in the query complexity, i.e., the number of queries required to conduct certain operations.

Using the inner ring as our index, and routing the shortest paths through it, we get small additive error and sublinear setup query cost: we also prove that our query cost per inquiry is subpolynomial (i.e., of the form $n^{o(1)}$, where the $o(1)$ term tends to zero as $n \to \infty$).

Theorem 4.5 (Subpolynomial query complexity for shortest paths). *Suppose that the preprocessing phase of our algorithm acquires C_{in}. The average query complexity of computing a path between a pair of vertices is bounded by $n^{\Theta(1/\log\log n)}$.*

Finally, we prove that preprocessing is necessary to achieve such query complexity per inquiry. We prove that any method for finding a path between two nodes u and v in a Chung-Lu random graph that does not employ preprocessing, requires $n^{\Omega(1)}$ node queries to succeed with good probability. Due to space considerations, we defer the proof sketch to Section 6 in the supplement; for a more complete proof of similar results for Erdős-Renyi and other random graphs, see Alon et al. [8].

Theorem 4.6 (Lower bound). *Fix $2 < \beta < 3$ and let $G \in \mathcal{G}_{CL}^{\beta}$ with average expected degree d (possibly depending on n, but satisfying $d = n^{o(1)}$). Further let $u \neq v$ be a random pair of nodes in G. Any algorithm that receives node query access to G starting at u and v must make, with high probability, $n^{\Omega(1)}$ queries to G in order to find a path between u and v.*

5 Experimental Results

In this section, we experimentally evaluate the performance of our algorithm. We look at several metrics to evaluate performance in different aspects. We compare with BiBFS, a traversal-based approach, and with the indexing algorithms PLL and MLL. We test several aspects, summarized next. Detailed results are provided in the rest of this section.

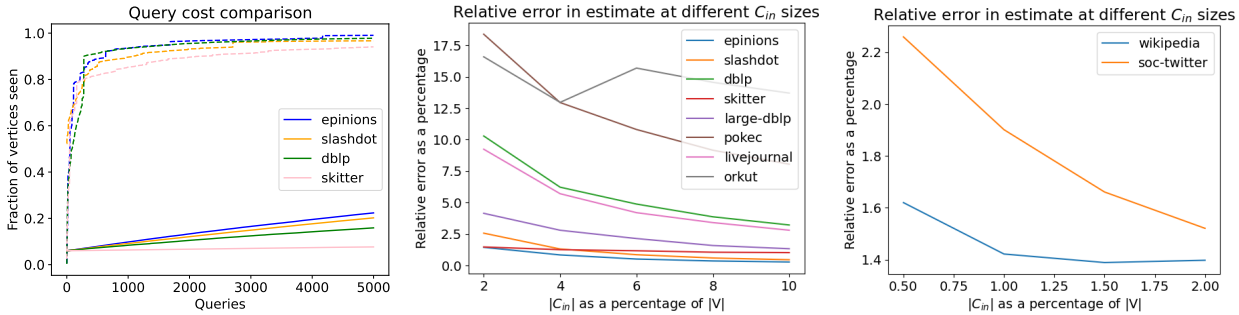
- **Query cost:** By query cost, we refer to the number of vertices queried by WormHole, consistent with our access model (see §1.2). We show that WormHole actually does remarkably well in terms of query cost, seeing a small fraction of the whole graph even for several thousands of shortest path inquiries. See Figures 2(b) and 4.
- **Inquiry time:** We demonstrate that $\texttt{WormHole}_E$ achieves consistent speedups over traditional BiBFS, even while using it as the sole primitive in the procedure. More complex methods such as PLL and MLL time out for the majority of large graphs. We also provide variants that achieve substantially higher speedups. Finally, in §5.3, we show how using the existing indexing-based state or the art methods on the core lets us achieve indexing-level inquiry times. See Figure 1.
- **Accuracy:** We show that our estimated shortest paths are accurate up to an additive error 2 on 99% of the inquiries for the default version $\texttt{WormHole}_E$; a faster heuristic, $\texttt{WormHole}_H$, shows lower accuracy, but still over 90% of inquiries satisfy this condition. See §5.1 and Table 1 for details.
- **Setup:** We look at the setup time and disk space for each method. Perhaps as expected, $\texttt{WormHole}_E$ beats the indexing based algorithms by a wide margin in terms of both space and time: see Figure 5. In §5.3 we further show that using these methods restricted to C_{in} results in a variant $\texttt{WormHole}_M$ with much lower setup cost (Table 4 in the supplement).

Datasets. The experiments have been carried out on a series of datasets of varying sizes, as detailed in Table 2. The datasets have been taken either from the SNAP large networks database [36] or the KONECT project [34]. We organize the results into two broad sections: we first introduce two variants of our algorithm. We then compare it with BiBFS as well as indexing based methods – PLL and MLL. The latter two did not terminate in 12 hours for most of the graphs, while BiBFS completed on even our largest networks.

We classify the examined graphs into three different classes and use a fixed percentage as the 'optimal' inner ring size for graphs of comparable size (where the inner core size as % of the total size decreases for larger networks, an indication for the sublinearity of our approach). This takes into account the tradeoff between accuracy and the query/memory costs incurred by a larger inner ring. The last column of Table 2 notes the sizes of lz. For the experimental section, we default to these sizes unless mentioned otherwise.

	BiBFS	WormHole$_E$					WormHole$_H$				
Network	MIT	MIT	SU/I	+0(%)	≤ +1 (%)	≤ +2 (%)	MIT	SU/I	+0(%)	≤ +1 (%)	≤ +2 (%)
epinions	144	41	4.5	98.06	99.99	100.00	20	24	66.97	99.54	100.00
slashdot	99	46	2.8	73.43	95.37	99.28	24	14	63.09	98.78	99.98
dblp	247	110	2.4	97.02	99.96	100.00	48	11	44.72	82.42	96.53
skitter	3004	1439	2.3	94.71	99.89	100.00	660	24	58.99	96.78	99.98
large-dblp	3041	1447	2.3	85.37	99.10	99.95	417	21	47.61	89.74	99.04
pokec	2142	1317	1.8	51.37	92.15	99.63	506	11	14.52	59.51	90.71
livejournal	8565	4318	2.1	71.98	97.95	99.86	1054	29	28.86	77.93	97.83
orkut	14k	3213	4.4	58.50	94.56	99.64	1030	35	20.66	68.11	93.93
wikipedia	35k	17k	2.4	94.94	99.92	100.00	3394	36	44.65	98.74	100.00
soc-twitter	204k	81k	3.4	93.30	99.98	100.00	12k	181	35.13	99.30	99.99

Table 1: Accuracy and speedup for the variants WormHole$_E$ and WormHole$_H$ (see §5.1). We note the mean inquiry times per inquiry (MIT) in microseconds, and average speed up *per inquiry* (SU/I) compared to BiBFS for each method. We also note the percentiles of inquiries by absolute additive error.



(a) Query cost of WormHole. (b) Mean rel. error with varying C_{in}. (c) Mean rel. error with varying C_{in} (large graphs).

Figure 4: (a) Query cost of WormHole and BiBFS on several datasets, measured as fraction of the graph seen by WormHole vs BiBFS over the first 5k inquiries in small and medium graphs (see Figure 1(b) for the results on large and huge graphs). The dotted lines refer to the query cost by BiBFS while the solid lines are due to WormHole. The latter two are accuracy of WormHole$_E$ in different settings across different datasets. We plot relative error at different inner ring sizes for the datasets.

Network	$\lvert V\rvert$	$\lvert E\rvert$	BiBFS	PLL	MLL	$\lvert C_{\text{in}}\rvert$
epinions	$7.6 \cdot 10^4$	$5.1 \cdot 10^5$	✓	✓	✓	6%
slashdot	$7.9 \cdot 10^4$	$5.2 \cdot 10^5$	✓	✓	✓	6%
dblp	$3.2 \cdot 10^5$	$1.0 \cdot 10^6$	✓	✓	✓	6%
skitter	$1.7 \cdot 10^6$	$1.1 \cdot 10^7$	✓	✓	✓	6%
large-dblp	$1.8 \cdot 10^6$	$2.9 \cdot 10^7$	✓	✓	✗	4%
soc-pokec	$1.6 \cdot 10^6$	$3.1 \cdot 10^7$	✓	✗	✗	4%
soc-live	$4.8 \cdot 10^6$	$6.8 \cdot 10^7$	✓	✗	✗	4%
soc-orkut	$3.1 \cdot 10^6$	$1.2 \cdot 10^8$	✓	✗	✗	4%
wikipedia	$1.4 \cdot 10^7$	$4.4 \cdot 10^8$	✓	✗	✗	1%
soc-twitter	$4.2 \cdot 10^7$	$1.5 \cdot 10^9$	✓	✗	✗	1%

Table 2: Network datasets used for experimental evaluation. The ✓ and ✗ symbols mark whether each method finished its run on the specified dataset within 12 hours.

Implementation details. We run our experiments on an AWS ec2 instance with 32 AMD EPYC™ 7R32 vCPUs and 64GB of RAM. The code is written in C++ and is available on github at https://github.com/Nadianw36/WormHole/. The backbone of the graph algorithms is a subgraph counting library that uses compressed sparse representations [45].

5.1 WormHole$_E$, WormHole$_H$ and BiBFS

We run two separate versions of WormHole: the one, as described in Algorithm 1, is what we refer to as WormHole$_E$ (exact). Another variant we consider is where we pick just the highest degree vertices from $C(s)$ and $C(t)$ respectively, and do a BFS from those. This cuts down on the inquiry time, but also reduces the accuracy. We call this variant WormHole$_H$. We take a deeper look into this tradeoff in the following sections. Note that both of these methods have the same query cost per shortest path inquiry, and the only difference is in running time (as all vertices in C_{in} have already been queried during CoreGen). For all runs, we do approximately 10,000 inquiries of uniformly chosen source and destination pairs (discarding disconnected pairs and other invalid inquiries). Our program runs on a compressed sparse representation (CSR) graph, and we store binary arrays for both C_{in} and C_{out} for practical purposes.

Query Cost. To examine query cost, we look at the number of vertices seen by the algorithm over the first 5000 inquiries and compare it to BiBFS. We direct the reader to Figure 2(b) for a summary of the query cost for our larger graphs, and Figure 4(a) for the same in the smaller ones. Consistently across all graphs, BiBFS quickly views between 70% and 100% of the vertices in just a few hundred inquiries. In comparison, the query cost of WormHole is

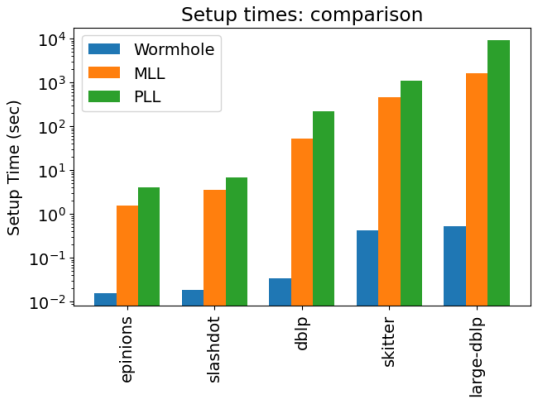


Figure 5: Comparison of setup times. The index-based methods did not terminate on graphs larger than these.

quite small for all networks: in the smaller networks, we see less than 30% of the vertices even after 5000 inquiries, and in the larger ones this number is less than 10% (in the largest ones, wikipedia and soc-twitter, it is <2%).

Accuracy. We consider two error measures, absolute (additive) and relative (multiplicative). Clearly, an additive error of, say, 2, is less preferable for shorter paths than for longer ones. The *relative error* $re(s,t)$ is more refined, as it measures the error with respect to the actual distance of the pair at question. Formally, $re(s,t) = \frac{\overline{d}(s,t) - d(s,t)}{d(s,t)}$, where for a vertex pair (s,t), $d(s,t)$ is the true distance and $\overline{d}(s,t)$ is the approximate distance estimated by WormHole. We investigate the relative error as a function of the core size – see Figure 4. In general, the accuracy drops as we decrease the size of the core. Moreover, we observe that larger graphs give comparably good results at much smaller inner ring sizes, keeping in line with our hypothesis of a sublinear inner ring.

Speedups over BiBFS *in inquiry time.* The main utility of $\texttt{WormHole}_H$ is in exhibiting efficiency at the expense of some accuracy; see Table 1. $\texttt{WormHole}_E$ already achieves speedups per inquiry over BiBFS. The variant $\texttt{WormHole}_H$ further speeds up each inquiry by another order of magnitude, up to a massive 181× in case of our largest network, soc-twitter (while utilizing the same decomposition and data structure). BiBFS does not have any setup cost, while WormHole does (for both variants). However, we show that our setup costs are typically very low: our highest setup time is about two minutes, and the highest setup space requirement is under 100MB. The complete statistics are provided in Table 3 in the supplement.

5.2 Comparison with index-based methods

As discussed, index construction allows for much faster inquiry times, but setup times that can take up to several hours even for relatively small-sized graphs. We attempt to benchmark our algorithm against two state of the art methods, PLL and MLL. PLL solves the easier task of finding distances, while MLL does explicit shortest path construction (the authors note that PLL may be extended to output paths, but no code is publicly available). However, once the graphs hit a few million vertices, these methods either take too long to run the setup, or even if they succeed in index construction, they may be too large to load into memory.

Mean inquiry time. In the cases where index based methods do succeed (graphs with fewer than 30 million edges), they have a clear advantage in per inquiry cost. Their typical inquiry time is in microseconds, where PLL is faster since it only computes distances, and MLL is about 3 times slower than PLL.

Setup cost. In terms of setup times, both $\texttt{WormHole}_E$ and $\texttt{WormHole}_H$ have a massive advantage over the index-based methods. Both PLL and MLL failed to complete setup for any graph with more than 30 million vertices in 12 hours. In comparison, even for our largest graph, soc-twitter (1.5 billion vertices), the setup time for $\texttt{WormHole}_E$ is just minutes. Even in the cases where these methods do terminate, the storage footprint is massive. We observed that if allowed to run, MLL completes index construction on soc-pokec in a little under 24 hours, but the constructed files are almost a combined 45 gigabytes in size; in comparison, the input COO file is only 250 megabytes! A detailed comparison of the space footprint is given in Figure 2 in §1, and the time comparisons can be found in the supplement.

5.3 WormHole as a primitive: $\texttt{WormHole}_M$

$\texttt{WormHole}_E$ and $\texttt{WormHole}_H$ perform well in terms of query cost and accuracy. The inquiry times, especially in the latter variant, are also huge improvements over BiBFS, but lag behind indexing based methods. However, as evident from our experiments, landmark based index creation may even be impossible for large networks. The success of $\texttt{WormHole}_E$ comes from exact shortest paths computed solely on a small core. In practice, a traversal based algorithm takes very little time to reach C_{in}; the bulk of the cost comes from the exact path computation inside C_{in}. How much faster can we make our algorithm if we speed this process up, perhaps by doing indexing solely on the core?

To this end, we propose as a third alternative, $\texttt{WormHole}_M$: it functions almost identical to $\texttt{WormHole}_H$, but instead of running BiBFS to find the shortest paths in C_{in}, it sets up MLL on all of C_{in} and then uses the MLL index to compute shortest paths inside C_{in}. This is an illustrative example to show how our decomposition can be used in combination with existing techniques. The accuracy guarantees of this variant as presented will be identical to $\texttt{WormHole}_H$; however, we do not analyze the index size theoretically.

We conduct similar experiments for $\texttt{WormHole}_M$. Remarkably, while MLL fails to complete setup on most of the graphs, $\texttt{WormHole}_M$ successfully runs it on the core in *all* cases. As noted in Table 4 in the supplement, the cost in both time and space is orders of magnitude smaller than for the full graph, though still significantly larger than the default $\texttt{WormHole}_H$. We note about two orders of magnitude of improvement in time per inquiry over $\texttt{WormHole}_H$ but are higher in both setup time and space by two orders of magnitude. Notably, even in cases where MLL does complete on the full graph, we are able to answer inquiries in roughly the same time (see Figure 1) at a fraction of the setup cost (Figure 5). We leave a more systematic investigation of this approach to future work.

References

[1] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. 2011. VC-Dimension and Shortest Path Algorithms. In *Automata, Languages and Programming (ICALP '11)*.

[2] Ittai Abraham, Daniel Delling, Amos Fiat, Andrew V. Goldberg, and Renato F. Werneck. 2016. Highway Dimension and Provably Efficient Shortest Path Algorithms. *J. ACM* 63, 5, Article 41 (2016).

[3] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. 2011. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *Experimental Algorithms*, Panos M. Pardalos and Steffen Rebennack (Eds.).

[4] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast Exact Shortest-Path Distance Queries on Large Networks by Pruned Landmark Labeling *(SIGMOD '13)*.

[5] Takuya Akiba, Christian Sommer, and Ken-ichi Kawarabayashi. 2012. Shortest-Path Queries for Complex Networks: Exploiting Low Tree-Width Outside the Core *(EDBT '12)*.

[6] Réka Albert, Hawoong Jeong, and Albert-László Barabási. 1999. Diameter of the World-Wide Web. *Nature* 401, 6749 (1999).

[7] André Allavena, Anirban Dasgupta, John Hopcroft, and Ravi Kumar. 2006. Finding (Short) Paths in Social Networks. *Internet Mathematics* 3, 2 (2006), 129 – 146.

[8] Noga Alon, Allan Grønlund, Søren Fuglede Jørgensen, and Kasper Green Larsen. 2023. Sublinear Time Shortest Path in Expander Graphs. arXiv:2307.06113 [cs.DS]

[9] Igor Artico, I Smolyarenko, Veronica Vinciotti, and Ernst C Wit. 2020. How rare are power-law networks really? *Proceedings of the Royal Society A* 476, 2241 (2020).

[10] Omri Ben-Eliezer, Talya Eden, Joel Oren, and Dimitris Fotakis. 2022. Sampling Multiple Nodes in Large Networks: Beyond Random Walks *(WSDM '22)*. 37–47.

[11] Thomas Bläsius, Cedric Freiberger, Tobias Friedrich, Maximilian Katzmann, Felix Montenegro-Retana, and Marianne Thieffry. 2022. Efficient Shortest Paths in Scale-Free Networks with Underlying Hyperbolic Geometry. *ACM Trans. Algorithms* 18, 2, Article 19 (mar 2022).

[12] Michele Borassi and Emanuele Natale. 2019. KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation. *ACM J. Exp. Algorithmics* 24, Article 1.2 (feb 2019).

[13] Arthur Brady and Lenore Cowen. 2006. Compact Routing on Power Law Graphs with Additive Stretch. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX '06)*.

[14] Flavio Chiericetti, Anirban Dasgupta, Ravi Kumar, Silvio Lattanzi, and Tamás Sarlós. 2016. On Sampling Nodes in a Network. In *Proceedings of the 25th International Conference on World Wide Web (WWW '16)*.

[15] Fan Chung and Linyuan Lu. 2002. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics* 6, 2 (2002).

[16] Fan Chung and Linyuan Lu. 2004. The average distance in a random graph with given expected degrees. *Internet Mathematics* 1, 1 (2004).

[17] Fan Chung and Linyuan Lu. 2006. The volume of the giant component of a random graph with given expected degrees. *SIAM Journal on Discrete Mathematics* 20, 2 (2006).

[18] Andrea Costa, Ana M. Martín González, Katell Guizien, Andrea M. Doglioli, José María Gómez, Anne A. Petrenko, and Stefano Allesina. 2019. Ecological networks: Pursuing the shortest path, however narrow and crooked. *Scientific Reports* 9, 1 (Nov. 2019).

[19] Mingyang Deng, Yael Kirkpatrick, Victor Rong, Virginia Vassilevska Williams, and Ziqian Zhong. 2022. New Additive Approximations for Shortest Paths and Cycles *(ICALP '22)*, Vol. 229.

[20] Jörg Derungs, Riko Jacob, and Peter Widmayer. 2007. Approximate Shortest Paths Guided by a Small Index. In *Algorithms and Data Structures, 10th International Workshop, WADS (Lecture Notes in Computer Science)*, Vol. 4619.

[21] E. W. Dijkstra. 1959. A Note on Two Problems in Connexion with Graphs. *Numer. Math.* 1, 1 (dec 1959).

[22] Michal Dory, Sebastian Forster, Yael Kirkpatrick, Yasamin Nazari, Virginia Vassilevska Williams, and Tijn de Vos. 2023. Fast 2-Approximate All-Pairs Shortest Paths. *CoRR* abs/2307.09258 (2023).

[23] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. 1999. On Power-Law Relationships of the Internet Topology *(SIGCOMM '99)*.

[24] John A Fitch III and Lance J Hoffman. 1993. A shortest path network security model. *Computers & Security* 12, 2 (1993), 169–189.

[25] Michael L. Fredman and Robert Endre Tarjan. 1987. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *J. ACM* 34, 3 (jul 1987).

[26] Peng Gao, Xusheng Xiao, Zhichun Li, Kangkook Jee, Fengyuan Xu, Sanjeev R. Kulkarni, and Prateek Mittal. 2019. A Query System for Efficiently Investigating Complex Attack Behaviors for Enterprise Security. *Proc. VLDB Endow.* 12, 12 (2019).

[27] Patrice Godefroid, Nils Klarlund, and Koushik Sen. 2005. DART: Directed Automated Random Testing. In *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation (PLD '05)*.

[28] Andrew V. Goldberg and Chris Harrelson. 2005. Computing the Shortest Path: A Search Meets Graph Theory *(SODA '05)*.

[29] Ruoming Jin, Zhen Peng, Wendell Wu, Feodor Dragan, Gagan Agrawal, and Bin Ren. 2020. Parallelizing Pruned Landmark Labeling: Dealing with Dependencies in Graph Algorithms *(ICS '20)*. Article 11.

[30] Xin Jin, Charalampos Katsis, Fan Sang, Jiahao Sun, Elisa Bertino, Ramana Rao Kompella, and Ashish Kundu. 2023. Prometheus: Infrastructure Security Posture Analysis with AI-generated Attack Graphs. *CoRR* (2023). arXiv:2312.13119

[31] Siddharth Kaza, Jennifer Xu, Byron Marshall, and Hsinchun Chen. 2009. Topological Analysis of Criminal Activity Networks: Enhancing Transportation Security. *IEEE Transactions on Intelligent Transportation Systems* 10, 1 (2009).

[32] Masahiro Kimura and Kazumi Saito. 2006. Tractable Models for Information Diffusion in Social Networks. In *Knowledge Discovery in Databases: PKDD 2006*.

[33] Maksim Kitsak, Alexander Ganin, Ahmed Elmokashfi, Hongzhu Cui, Daniel A. Eisenberg, David L. Alderson, Dmitry Korkin, and Igor Linkov. 2023. Finding shortest and nearly shortest path nodes in large substantially incomplete networks by hyperbolic mapping. *Nature Communications* 14, 1 (2023).

[34] Jérôme Kunegis. 2013. KONECT: The Koblenz Network Collection *(WWW '13 Companion)*.

[35] Andrey Kutuzov, Mohammad Dorgham, Oleksiy Oliynyk, Chris Biemann, and Alexander Panchenko. 2019. Learning Graph Embeddings from WordNet-based Similarity Measures. In *Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (*SEM 2019)*.

[36] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[37] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2020. Scaling Up Distance Labeling on Graphs with Core-Periphery Properties *(SIGMOD '20)*.

[38] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2020. Scaling up distance labeling on graphs with core-periphery properties *(SIGMOD '20)*. 1367–1381.

[39] Wentao Li, Miao Qiao, Lu Qin, Ying Zhang, Lijun Chang, and Xuemin Lin. 2022. Distance labeling: on parallelism, compression, and ordering. *The VLDB Journal* 31 (01 2022).

[40] Ye Li, Leong Hou U, Man Lung Yiu, and Ngai Meng Kou. 2017. An Experimental Study on Hub Labeling Based Shortest Path Algorithms. *Proc. VLDB Endow.* 11, 4 (dec 2017).

[41] Linyuan Lincoln Lu. 2002. *Probabilistic methods in massive graphs and Internet computing*. Ph.D. Dissertation. University of California, San Diego.

[42] Amgad Madkour, Walid G Aref, Faizan Ur Rehman, Mohamed Abdur Rahman, and Saleh Basalamah. 2017. A survey of shortest-path algorithms. *arXiv preprint arXiv:1705.02044* (2017).

[43] Fragkiskos D. Malliaros, Christos Giatsidis, Apostolos N. Papadopoulos, and Michalis Vazirgiannis. 2019. The Core Decomposition of Networks: Theory, Algorithms and Applications. *The VLDB Journal* 29, 1 (nov 2019).

[44] M. E. J. Newman. 2003. The Structure and Function of Complex Networks. *SIAM Rev.* 45, 2 (2003).

[45] Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. 2017. ESCAPE: Efficiently Counting All 5-Vertex Subgraphs *(WWW '17)*.

[46] Ira Pohl. 1969. *Bi-directional and heuristic search in path problems*. Ph.D. Dissertation. Stanford University, USA.

[47] Jianzhong Qi, Wei Wang, Rui Zhang, and Zhuowei Zhao. 2020. A Learning Based Approach to Predict Shortest-Path Distances *(EBDT'20)*.

[48] Fatemeh Salehi Rizi, Joerg Schloetterer, and Michael Granitzer. 2020. Shortest Path Distance Approximation Using Deep Learning Techniques *(ASONAM '18)*.

[49] Liam Roditty. 2023. New Algorithms for All Pairs Approximate Shortest Paths *(STOC '23)*.

[50] M. Puck Rombach, Mason A. Porter, James H. Fowler, and Peter J. Mucha. 2014. Core-Periphery Structure in Networks. *SIAM J. Appl. Math.* 74, 1 (2014).

[51] Barna Saha and Christopher Ye. 2023. Faster Approximate All Pairs Shortest Paths. *CoRR* abs/2309.13225 (2023).

[52] Stephen B. Seidman. 1983. Network structure and minimum degree. *Social Networks* 5, 3 (1983).

[53] Sandeep Sen. 2009. Approximating shortest paths in graphs. In *International Workshop on Algorithms and Computation*. 32–43.

[54] Matteo Serafino, Giulio Cimini, Amos Maritan, Andrea Rinaldo, Samir Suweis, Jayanth R. Banavar, and Guido Caldarelli. 2021. True scale-free networks hidden by finite size effects. *Proceedings of the National Academy of Sciences* 118, 2 (2021).

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

[56] Chi Wang, Wei Chen, and Yajun Wang. 2012. Scalable Influence Maximization for Independent Cascade model in Large-scale Social Networks. *Data Mining and Knowledge Discovery Journal* 25, 3 (2012).

[57] Jennifer J Xu and Hsinchun Chen. 2004. Fighting organized crimes: using shortest-path algorithms to identify associations in criminal networks. *Decision Support Systems* 38, 3 (2004).

[58] Xiaolong Xu, Yuancheng Li, Tao Huang, Yuan Xue, Kai Peng, Lianyong Qi, and Wanchun Dou. 2019. An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks. *Journal of Network and Computer Applications* (2019).

[59] Zhiqiang Xu, Pengcheng Fang, Changlin Liu, Xusheng Xiao, Yu Wen, and Dan Meng. 2022. DEPCOMM: Graph Summarization on System Audit Logs for Attack Investigation. In *IEEE Symposium on Security and Privacy (SP)*.
[60] Hongyang Zhang, Huacheng Yu, and Ashish Goel. 2019. Pruning Based Distance Sketches with Provable Guarantees on Random Graphs *(WWW '19)*.
[61] Junhua Zhang, Wentao Li, Long Yuan, Lu Qin, Ying Zhang, and Lijun Chang. 2022. Shortest-Path Queries on Complex Networks: Experiments, Analyses, and Improvement. *Proc. VLDB Endow.* 15, 11 (2022).
[62] Xiao Zhang, Travis Martin, and M. E. J. Newman. 2015. Identification of core-periphery structure in networks. *Physical Review E* 91, 3 (mar 2015).
[63] Xiaohan Zhao, Alessandra Sala, Christo Wilson, Haitao Zheng, and Ben Y. Zhao. 2010. Orion: Shortest Path Estimation for Large Social Graphs *(WOSN'10)*.
[64] 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)*.
[65] Uri Zwick. 2001. Exact and Approximate Distances in Graphs — A Survey. In *Algorithms — ESA 2001*.
[66] Uri Zwick. 2002. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM (JACM)* 49, 3 (2002), 289–317.