Shortest Paths Discovery in Uncertain Networks via Transfer Learning

SHIXUN HUANG, RMIT University, Australia
ZHIFENG BAO*, RMIT University, Australia

Due to various reasons such as noisy measurement and privacy preservation, a network/graph is often uncertain such that each edge in the network has a probability of existence. In this paper, we study finding the most probable shortest path which has the highest probability of being the shortest path between a given pair of nodes in an uncertain network. Despite significant progress being made, this problem still suffers from the efficiency and scalability issue. To solve this problem, the state-of-the-art adopts a two-phase approach where Phase 1 generates some candidate paths and Phase 2 estimates their probabilities of being the shortest path and returns the one with the highest probability as the solution. Notably, Phase 2 requires a large number of simulations over all edges in the network and can easily dominate the cost of the whole process. In this paper, we aim to resolve the efficiency and scalability issue by optimizing Phase 2. Specifically, we first propose a non-learning based fast approximation technique which significantly reduces the number of samples for the probability estimation in each simulation. Afterwards, we further propose a learning-based method which can directly estimate the probability of each candidate path without costly simulations. Extensive experiments show that (1) compared to the state-of-the-art, our fast approximation technique and learning-based method can achieve up to 5x and 210x speedups in Phase 2 respectively while maintaining highly competitive or even equivalent results, (2) the training process is highly scalable and (3) the prediction function can work effectively under the problem settings different from the one it was trained.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Theory of computation** → *Design and analysis of algorithms.*

Additional Key Words and Phrases: shortest path, uncertain networks, transfer learning

ACM Reference Format:
Shixun Huang and Zhifeng Bao. 2023. Shortest Paths Discovery in Uncertain Networks via Transfer Learning. *Proc. ACM Manag. Data* 1, 2, Article 141 (June 2023), 25 pages. https://doi.org/10.1145/3589286

1 INTRODUCTION

Graphs are often found in important and emerging domains, including traffic networks, biological networks, and sensor networks. These graphs are inherently uncertain in many cases due to various reasons including but not limited to noisy measurements [2], hardware limitation [3], inference models [23], and privacy-preserving perturbation [6]. To model the uncertainty, uncertain graphs, where each edge is associated with a probability of existence, have been studied extensively in many problems such as motif discovery [44], k-nearest neighbor queries [43], reachability

*The corresponding author.

Authors' addresses: Shixun Huang, shixun.huang@rmit.edu.au, RMIT University, Melbourne, Australia; Zhifeng Bao, zhifeng.bao@rmit.edu.au, RMIT University, Melbourne, Australia.

2836-6573/2023/6-ART141 $15.00
https://doi.org/10.1145/3589286

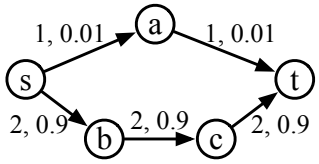


Fig. 1. A simple uncertain graph

queries [40], clustering [29], sampling [49], network design [39], influence maximization [34] and embedding [30].

In this paper, we study the problem of finding the Most Probable Shortest Path in uncertain networks (MPSP). Unlike in deterministic networks where the path length is the only factor to consider, we should consider both the length and the existence probability of paths for discovering the shortest paths. Specifically, given an uncertain network $\mathcal{G}$, a source node s and a destination node t, we aim to find the most probable path MPSP(s, t) which has the highest probability of being the shortest path from s to t, i.e., the probability that MPSP(s, t) exists and no path shorter than MPSP(s, t) exists in the uncertain network.

Example 1. *Figure 1 shows a simple uncertain network where the first and second numbers associated with each edge denote the edge length and existence probability respectively. We have two paths from s to t, namely path P_1 via node a and path P_2 via nodes b and c. Even though P_1 has a smaller length than P_2, P_1 is not the MPSP in this uncertain network. Specifically, the existence probability of P_1 is 10^{-4} whereas the probability that P_2 exists but P_1 does not exist (i.e., P_2 is the shortest path) is $0.9^3 \times (1 - 10^{-4}) = 0.73$. Therefore, P_2 is the MPSP in this uncertain network.*

Shortest-path queries [7, 22, 37] are one of the most important graph primitives [53] and finding the MPSPs can be very useful in many applications. In road networks which can be modeled as uncertain graphs due to unexpected traffic jams or blockage, drivers may need to find the MPSPs to reach the destinations with the smallest amount of time [10, 11, 31, 58]. In sensor networks where links between sensor nodes have failure probabilities, finding the MPSPs can be helpful in routing between sensors [27]. In brain networks where nodes refer to the brain regions of interest (ROI), edges refer to potential co-activation between ROIs, and edge probability indicates the strength of the co-activation signal [13], finding the MPSPs can be useful in distinguishing between healthy and unhealthy brains with diseases such as autism [19, 26].

Finding the MPSP is very challenging since computing the probability of a path being the shortest path is #P-hard [53] and we need to find the path with the highest probability from considerable paths between s and t. Despite significant efforts being made [53, 61], the efficiency issue is still the bottleneck of this problem. The state-of-the-art [53] adopts a two-phase approach. In Phase 1, candidate paths are generated by combining the Dijkstra algorithm with Monte Carlo simulations. In Phase 2, the probability of each candidate path being the shortest path is computed and the one with the highest probability is returned as the solution. Specifically, to compute the probability of each candidate P, a number of simulations are required. In each simulation, every edge in the graph will be sampled to check if there exists a candidate path shorter than P.

In this paper, we focus on optimizing Phase 2 which accounts for a notable portion of the total running time and can easily dominate the total cost. We make two observations and correspondingly propose two approaches to boost the efficiency.

Observation 1: No need to perform sampling for all edges and evaluate the condition only after the sampling process. Instead, we only need to sample edges in candidate paths and this process can be terminated early if we conduct the sampling process and condition evaluation

simultaneously. Thus, we propose a fast probability approximation approach which replaces the original Phase 2 and achieves notable empirical speedups.

Observation 2: The number of simulations cannot be ignored and the time cost of each simulation can still be very high in the worst case. Thus, we propose a learning-based method which predicts the expected value (i.e., the number of times the condition is satisfied) of the aforementioned condition without costly simulations. We find that directly training a prediction function specific to our problem is not feasible since our prediction problem is a very special case of a general prediction problem and generating training data for this specific problem is very costly. To alleviate the efficiency issue and better generalize the predictive power, we resort to train a prediction function on a general problem where the training data is much easier to generate, and then transfer the trained prediction function to solve our prediction problem.

Our contributions are summarized as below.

- We propose a novel fast probability approximation technique which efficiently estimates the probability of each candidate path being the shortest path by performing the sampling process and the condition evaluation simultaneously, which notably reduces the sampling size and boosts the estimation process. (Section 3)
- We formulate the MPSP as a prediction problem, train a simple yet effective prediction function on a more general prediction problem beneficial to fast training data generation and generalization of the prediction function, and then transfer the trained function to solve our own prediction problem. (Section 4)
- Our extensive experiments on large-scale real-world datasets demonstrate that (i) compared to the state-of-the-art [53], our fast probability approximation method and learning-based method achieve up to 5x and 210x speedups in Phase 2 respectively while maintaining highly competitive results, (ii) the training process is highly scalable and the training cost is very small such that it can be ignored when compared with the total cost of answering all potential queries, and (iii) our trained prediction function is highly general and robust such that it can produce highly effective results under settings different from the one where it was trained. (Section 5)

2 PROBLEM FORMULATION AND BACKGROUND ON STATE-OF-THE-ART

We denote an uncertain graph/network as $\mathcal{G} = (V, E, l, p)$ where V and E refer to the set of nodes and edges respectively, $l : e \in E \rightarrow \mathbb{R}_{\geq 0}$ defines the length of each edge and $p : e \in E \rightarrow (0, 1]$ assigns an existence probability to each edge. Following existing literature on uncertain graphs [5, 32, 36, 50, 55, 58, 61], we interpret an uncertain graph $\mathcal{G}$ as a probability distribution over the $2^{|E|}$ possible worlds. Each possible world is a *deterministic* graph $G = (V, E_G, l) \sqsubseteq \mathcal{G}$ and it is obtained by sampling each edge $e \in E$ independently at random with probability $p(e)$. Specifically, a possible world G can be constructed by removing each edge e with probability $1 - p(e)$. Thus, the probability of observing the possible world $G = (V, E_G, l)$ is:

$$Pr(G) = \prod_{e \in E_G} p(e) \prod_{e \in E \setminus E_G} (1 - p(e)).$$

For a pair of distinct nodes s and t in V, we define a simple path P from s to t as an ordered sequence/set of edges, i.e., $P = \{e_1, e_2, ..., e_{|P|}\}$ where (i) $\forall e_i = (u_i, v_i)$ and $e_{i+1} = (u_{i+1}, v_{i+1})$ $(1 \leq i < |P|), v_i = u_{i+1}$, (ii) $u_0 = s$ and $v_{|P|} = t$ and (iii) $\forall i, j$ $(1 \leq i \neq j \leq |P|)$, $u_i \neq u_j$. The length $l(P)$ of path P is defined as the sum of lengths of each edge in P, i.e., $l(P) = \sum_{i=1}^{|P|} l(e_i)$. We denote $\mathcal{SP}(G, s, t)$ as the set of all shortest paths from s to t in the possible world G.

We denote the event that a path P exists in $\mathcal{G}$ by $\mathcal{X}(P)$ and $Pr(\mathcal{X}(P)) = \prod_{i=1}^{|P|} p(e_i)$. Correspondingly, we use $\overline{\mathcal{X}(P)}$ to denote the event that a path P does not exist in $\mathcal{G}$ and $Pr(\overline{\mathcal{X}(P)}) = 1 - Pr(\mathcal{X}(P))$.

We use $SP_s^t(P)$ to denote the event that P happens to be the shortest path from s to t in $\mathcal{G}$. Specifically, its probability is computed as below:

$$Pr(SP_s^t(P)) = \sum_{G \sqsubseteq \mathcal{G}} Pr(G) \times \mathbb{1}[P \in \mathcal{SP}(G, s, t)]$$

where $\mathbb{1}[\cdot]$ is an indicator function.

Definition 1 (Most Probable Shortest Path (MPSP)). *Given an uncertain graph $\mathcal{G} = (V, E, l, p)$ and two nodes $s, t \in V$, we aim to find the MPSP(s, t) which is the shortest path from s to t in $\mathcal{G}$ with the highest probability, i.e.,*

$$MPSP(s, t) = \underset{P \in \mathcal{P}(\mathcal{G}, s, t)}{\arg\max}\ Pr(SP_s^t(P)) \tag{1}$$

where $\mathcal{P}(\mathcal{G}, s, t)$ denotes the set of all paths from s to t in $\mathcal{G}$.

Solution Framework and Our Focus. Saha et al. [53] propose the state-of-the-art two-phase framework via a combination of existing techniques to solve this problem. Specifically, they adopt the Dijkstra+Monte Carlo simulations (Dijkstra+MC) [10] to generate some candidate paths in Phase 1, and then adopt the Luby-Karp algorithm [38] to approximate $Pr(SP_s^t(\cdot))$ of each candidate path and return the path with the highest probability as the result. They also show that this framework can provide good theoretical guarantees that the MPSP can be returned with a high probability [53].

In this paper, we focus on optimizing the efficiency of Phase 2 since it accounts for a large portion of the total cost, and it can easily dominate the whole cost as illustrated in the introduction and indicated in later time complexity analysis and experiments. Therefore, we will introduce these two phases separately and focus on introducing the framework and Phase 1 in this section.

Algorithm 1: Approximate MPSP(s, t) Discovery [53]

Input : An uncertain graph $\mathcal{G} = (V, E, l, p)$, a source node s, a target node t, positive integers m and N.
Output: An approximate MPSP(s, t).

1 $CP \leftarrow \emptyset$;
// Phase 1: Lines 2 to 4;
2 **for** $iter = 1$ *to* m **do**
3 $\quad P \leftarrow \mathsf{CPG}(\mathcal{G}, s, t)$;
4 $\quad$ **if** $|P| > 0$ **then** $CP \leftarrow CP \cup \{P\}$;
// Phase 2: Lines 5 to 8;
5 Order all paths in CP in increasing order of length;
6 **for** $j = 1$ *to* $|CP|$ **do**
7 $\quad \widetilde{Pr}(SP_s^t(CP[j])) \leftarrow \mathsf{PA}(\mathcal{G}, s, t, CP[j], \{CP[1], \ldots, CP[j-1]\}, N)$;
8 **return** $\arg\max_{P \in CP} \widetilde{Pr}(SP_s^t(P))$;

Algorithm 1 describes the two-phase framework. In Phase 1, the Candidate Path Generation (CPG) (Algorithm 2) is executed m rounds and in each round CPG tries to generate a candidate path. In Phase 2, the approximate probability $\widetilde{Pr}(SP_s^t(P))$ of each candidate path P being the shortest path is computed and the path with the highest probability is returned. Below is Phase 1 detail.

Given the source and target pair s-t, to sample a candidate path, the most straightfoward approach is to sample a possible world first and then call the Dijkstra algorithm to return the shortest path between s and t in this possible world as the result. However, this approach will inevitably incur a significant number of unnecessary samplings since many sampled edges are not relevant to the

Algorithm 2: Candidate Path Generation (CPG) [53]

Input : An uncertain graph $\mathcal{G} = (V, E, l, p)$, a source node s and a target node t.
Output : A candidate path to be MPSP(s, t).

$u \leftarrow s, CK \leftarrow \{s\}, Dis(s, s) \leftarrow 0, Dis(s, v) \leftarrow \infty \; \forall v \in V \setminus \{s\}, Seq(s, v) \leftarrow \emptyset \; \forall v \in V$;
repeat
  foreach $e = (u, v) \in E$ *where* $v \notin CK$ **do**
    if $Dis(s, v) > Dis(s, u) + l(e)$ **then**
      With probability $p(e)$, $Seq(s, v) \leftarrow Seq(s, u) \cup \{e\}$;
  $u \leftarrow \arg\min_{v \in V \setminus CK} Dis(s, v)$;
  $CK \leftarrow CK \cup \{u\}$;
until $u = t$ *or* $Seq(s, u) = \emptyset$;
return $Seq(s, t)$

Algorithm 3: Probability Approximation (PA) [53]

Input : An uncertain graph $\mathcal{G} = (V, E, l, p)$, candidate paths from s to t: P and $\{P_1, \ldots, P_n\}$ shorter than P, and an integer N.
Output : Approximation of $Pr(SP_s^t(P))$.

$Cnt \leftarrow 0, Z \leftarrow \sum_{i=1}^{n} Pr(\mathcal{X}(P_i \setminus P))$;
for $iter = 1$ *to* N **do**
  Sample $i \in \{1, 2, .., n\}$ with probability $\frac{Pr(\mathcal{X}(P_i \setminus P))}{Z}$;
  Sample a possbile world $G = (V, E_G, l) \sqsubseteq \mathcal{G}$ such that $(P_i \setminus P) \cup P = (P_i \cup P) \subseteq E_G$;
  if $\forall 1 \leq j < i, (P_j \setminus P) \nsubseteq E_G$ **then**
    $Cnt \leftarrow Cnt + 1$;
$\widetilde{p} \leftarrow \frac{Cnt}{N} \cdot Z$;
return $(1 - \widetilde{p}) \cdot Pr(\mathcal{X}(P))$

candidate path. To mitigate this issue, in Phase 1, Saha et al. [53] perform the Dijkstra algorithm and edge sampling simultaneously, and terminates the sampling process once the candidate path is found. Thus, this strategy is also called Dijkstra+Monte Carlo simulations (Dijkstra+MC). The rationale of Dijkstra+MC is similar to the traditional Dijkstra algorithm in deterministic graphs. The only notable difference is that when a node is reached in an uncertain graph, outgoing edges need to be sampled based on their probabilities. Algorithm 2 describes this process where CK is used to record nodes to which the shortest paths have been found, $Dis(s, v)$ is used to store the distance between s and v, and $Seq(s, v)$ is used to store a path (i.e., an ordered sequence of edges) from s to v.

3 NON-LEARNING BASED OPTIMIZATION

In this section, we will introduce the original Phase 2 (proposed in [53]) followed by our non-learning based optimization.

Original Phase 2: Probability Approximation (PA). In this phase, the probability of each path being the shortest path in the original uncertain graph is approximated by the Luby-Karp algorithm [38]. The intuition of this algorithm is that: given a path P, a set of candidate paths $\{P_1, \ldots, P_n\}$ shorter than P and a positive integer N, it first approximates the probability $\widetilde{p}$ of candidate paths shorter than P appearing in the input uncertain graph based on N possible worlds. Specifically, in each possible world, an integer i $(1 \leq i \leq n)$ is sampled to make sure that edges in $P_i \cup P$ exist in this world, the existence of all other edges is decided based on the sampling, and we need to check whether there exist candidate paths shorter than P_i in this world. The goal of sampling

Algorithm 4: Fast Probability Approximation (FPA)

Input : An uncertain graph $\mathcal{G} = (V, E, l, p)$, candidate paths from s to t: P and $\{P_1, \ldots, P_n\}$ shorter than P, and an integer N.
Output : Estimation of $Pr(SP_s^t(P))$.

1 $Cnt \leftarrow 0, Z \leftarrow \sum_{i=1}^{n} Pr(\mathcal{X}(P_i \setminus P))$;
2 **for** $iter = 1$ *to* N **do**
3 Sample $i \in [1, n]$ with probability $\frac{Pr(\mathcal{X}(P_i \setminus P))}{Z}$;
4 $Cnt \leftarrow Cnt+$ SET$(P, \{P_1, ..., P_i\})$;
5 $\widetilde{p} \leftarrow \frac{Cnt}{N} \cdot Z$;
6 **return** $(1 - \widetilde{p}) \cdot Pr(\mathcal{X}(P))$;

Algorithm 5: Sampling with Early Termination (SET)

Input : Path P, and paths $\{P_1, \ldots, P_i\}$ shorter than P.
Output : 1 or 0.

1 $G \leftarrow P_i \cup P$; $CK \leftarrow P_i \cup P$;
2 **for** $j = 0$ *to* $i - 1$ **do**
3 $Terminate \leftarrow True$;
4 **foreach** e *in* $P_j \setminus P$ **do**
5 **if** $e \notin CK$ **then**
6 Add e into CK;
7 Add e into G with the probability of $p(e)$;
8 **if** $e \notin G$ **then**
9 $Terminate \leftarrow False$; Break;
10 **if** $Terminate == True$ **then**
11 **return** 0
12 **return** 1

possible worlds is to estimate the probability of each P_i being the shortest path and subsequently compute $\widetilde{p}$. Afterwards, the value of $(1-\widetilde{p}) \cdot Pr(\mathcal{X}(P))$ is returned as an approximation of $Pr(SP_s^t(P))$. Algorithm 3 describes the overall process. Ideally, to approximate $Pr(SP_s^t(P))$, all paths shorter than P should be considered in this algorithm. However, this idea is infeasible and not scalable in practice. It is shown that using the generated candidate paths only can provide good theoretical guarantees that the MPSP can be returned with a high probability [53].

Time Complexity Analysis. In Phase 1, the Dijkstra algorithm is executed for m rounds, which incurs $O(m(|E| + |V| \log |V|))$ time. In Phase 2, candidate paths need to be sorted first, which requires $O(m \log m)$ time. Algorithm 3 will be executed m times and each run will cost $O(N|E|)$ time. Therefore, the total time complexity of Algorithm 1 is $O(m(N|E| + |V| \log |V| + \log m))$.

The Efficiency Issue. The probability approximation (i.e., Algorithm 3) suffers from a serious scalability issue since it incurs $O(N|E|)$ time and N is usually not small (e.g., 1000 in the experiments of [53]). Notably, this phase can easily dominate the whole cost for solving multiple queries, sharing the same source or destination nodes, at the same time. Specifically, for such queries, we can run Dijkstra+MC to generate candidates for different query pairs at the same time, whereas Phase 2 needs to run separately for each query. We find that there are some optimization techniques to improve the efficiency (e.g., 5x speedups in our experiments) by answering the follow two questions.

Q1: Do we need to sample the whole possible world? The answer is no. The reason is that not all edges in $\mathcal{G}$ are relevant to the condition in Line 5 which checks whether there exists a path P_j shorter than P_i in this possible world. Instead, we only need to sample the partial possible world

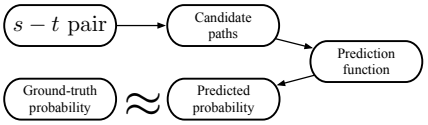


Fig. 2. A sketch of conceptual training

which only consists of edges in candidate paths shorter than P. This strategy is quite effective since the size (bounded by m) of candidate paths is small in practice.

Q2: Do we need to perform sampling and condition check separately? The answer is no. We observe that this condition will fail once there exists a P_j such that $(P_j \setminus P) \subseteq E_G$. Thus, if we perform sampling for edges in $(P_j \setminus P)$ first, we can quickly know that $(P_j \setminus P) \subseteq E_G$ and thus do not increment Cnt without sampling edges in other candidate paths.

Optimized Phase 2: Fast Probability Approximation (FPA). Even though we do not know which P_j will fail to meet this condition, this observation inspires us to (1) treat each candidate path as a group and perform sampling for edges in groups, instead of randomly deciding the order of edges for sampling, and (2) conduct the condition check simultaneously. Algorithm 4 describes this idea which combines the possible world sampling and condition checking in Algorithm 5. In Algorithm 5, each loop j corresponds to sampling edges in a candidate path P_j, G is used to store edges in the possible world and CK is used to record edges already sampled to avoid repetitive sampling for the same edge. If there exists a P_j such that all edges in $(P_j \setminus P)$ appear in G, the variable $Termination$ will be set as $True$ and 0 will returned. Otherwise, 1 will be returned.

Time Complexity Analysis. The time complexity of Algorithm 4 is $O(N \cdot \sum_{P \in CP} |P|)$ which is notably smaller than the time complexity $O(N|E|)$ of Algorithm 3 in practice. Moreover, the early termination technique in Algorithm 5 further boosts Algorithm 4.

4 LEARNING BASED OPTIMIZATION

Despite that we have introduced novel non-learning based optimization techniques to improve the efficiency of Phase 2, the efficiency is still limited by the large number N of simulations for probability estimation. In this section, we aim to propose a learning-based solution to compute the probability of each candidate path being the shortest path without running costly simulations. In what follows, we will introduce a conceptual training sketch (Section 4.1) which is infeasible in practice and lacks of generality but motivates us to design a practical training process. Then, in Section 4.2, we will show that our prediction problem can be formulated as the **Candidate-path-based Problem** which aims to predict candidate path existence probability. However, directly solving this prediction problem is infeasible since generating training data is very costly. Thus, we resort to solve a more general prediction problem called the **Random-walk-based Problem**, which aims to predict random-walk path existence probability such that the data generation process is much more efficient and generalization power can be greatly improved. Then, the practical learning process for the Random-walk-based Problem is proposed (Section 4.3) and the trained prediction function is transferred to solve the Candidate-path-based Problem (Section 4.4).

4.1 Conceptual Training Sketch

Figure 2 shows the straightforward training procedure to train a prediction function to replace the non-learning based methods for candidate paths' probability estimation. First, candidate paths are generated based on the s-t pair. Next, the prediction function takes the candidate paths as the input to predict the probability of each path. Last, it aims to minimize the difference between the ground-truth probability and the predicted one.

Algorithm 6: Learning-based Solution Sketch

Input : An uncertain graph $\mathcal{G} = (V, E, l, p)$ and candidate paths from s to t: P and $\{P_1, ..., P_n\}$ shorter than P.

Output : Estimation of $Pr(SP_s^t(P))$.

```
1 p̃ ← 0;
2 for i = 1 to n do
3 |   P̃r(E_i) ← EventProbabilityPrediction({P_1, ..., P_i});
4 |   p̃ ← p̃ + Pr(X(P_i \ P)) · P̃r(E_i);
5 return (1 − p̃) · Pr(X(P));
```

However, this training procedure is impractical since the number of training pairs might be large and we need to execute Dijkstra+MC for m rounds to generate candidate paths for every single training s-t pair. Worse still, the low-level design of this procedure and parameter tuning may require multiple runs of the whole process. Pre-storing all training pairs together with their candidate paths is not an effective remedy since such a strategy will incur significant storage usage. In the worst case, candidate paths of one training pair may cover all edges in the original graph. Furthermore, the training input (i.e., an s-t pair) is too specific such that there is a lack of an intuitive way to decide which pairs to train such that the prediction function can give accurate probability estimation for candidates paths of unseen test pairs.

Therefore, to improve the learning efficiency and robustness of the prediction function, an advanced training procedure design is required to mitigate: **the efficiency issue** brought by candidate paths generation and **the generality issue** brought by the over-specific training input and prediction.

4.2 The Two Problems and the Workflow

In this section, we will introduce the Candidate-path-based Problem and how it can help solve the MPSP problem. Then, we will introduce the more general Random-walk-based Problem and discuss how solving it helps alleviate the aforementioned issues without costly executions of the Dijkstra+MC algorithm.

4.2.1 The Candidate-path-based Problem. To facilitate the description, we need to simplify the rationale of the Luby-Karp idea as in Algorithm 3. In expectation, the index i will be sampled $N \cdot \frac{Pr(\mathcal{X}(P_i \setminus P))}{Z}$ times. When index i is sampled, edges in $(P_i \setminus P) \cup P = P_i \cup P$ are assumed to exist and other edges in the uncertain graph will be sampled to form a possible world. In the possible world, we check whether there exists any path shorter than P_i. If the answer is 'no', we add Cnt by 1. Thus, the contribution from P_i to Cnt is the number of times we have a 'no' answer when the index i is sampled (i.e., edges in $P_i \cup P$ are assumed to exist).

Let E_i denote the event that, given P_i exists, no candidate paths shorter than P_i exist in the uncertain graph. In expectation, where index i is sampled, the contribution from P_i to Cnt is equal to the probability $Pr(E_i)$ of E_i. It can be expressed as:

$$Pr(E_i) = Pr(\bigcap_{j=1}^{i-1} \overline{\mathcal{X}(P_j \setminus P)} \mid \mathcal{X}(P_i \setminus P)). \tag{2}$$

Given the final $Cnt = \sum_{i=1}^{n} N \frac{Pr(\mathcal{X}(P_i \setminus P))}{Z} Pr(E_i)$, $\widetilde{p}$ can be calculated as:

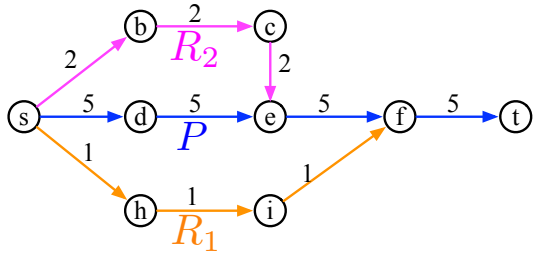


Fig. 3. A simple graph with some paths in different colors

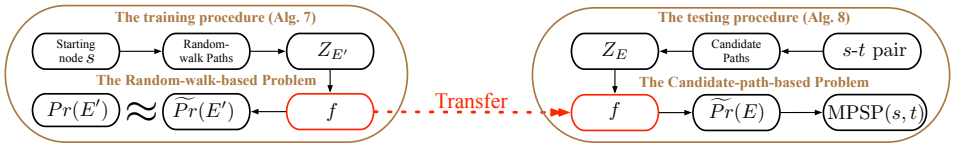


Fig. 4. The whole workflow of our learning-based solution

$$\widetilde{p} = Cnt/N \cdot Z = \left(\sum_{i=1}^{n} N \cdot \frac{Pr(\mathcal{X}(P_i \setminus P))}{Z} Pr(E_i)\right)/N \cdot Z$$
$$= \sum_{i=1}^{n} Pr(\mathcal{X}(P_i \setminus P)) \cdot Pr(E_i).$$

What to predict. Since $Pr(\mathcal{X}(P_i \setminus P))$ can be easily computed, we aim to predict a value $\widetilde{Pr}(E_i)$ to approximate $Pr(E_i)$ such that $\widetilde{p}$ and the subsequent $Pr(SP_s^t(P)) = (1-\widetilde{p}) \cdot Pr(\mathcal{X}(P))$ can be computed. Algorithm 6 describes the sketch of this process.

4.2.2 The Random-walk-based Problem. To avoid generating candidate paths via Dijkstra+MC, we study a more general prediction problem (Problem B). In this problem, we are given a set $\mathcal{S}$ of random-walk paths $R_1, R_2, ..., R_{|\mathcal{S}|}$ generated by independent random walks which start from the same node, and a target random-walk path R_i $(1 \le i \le |\mathcal{S}|)$ from $\mathcal{S}$. The objective is to predict the probability $Pr(E_i')$ of the event E_i' that, given R_i exists, other paths in $\mathcal{S}$ do not exist in this uncertain graph. Formally, it can be expressed as below:

$$Pr(E_i') = Pr(\bigcap_{R_j \in \mathcal{S} \wedge j \neq i} \overline{\mathcal{X}(R_j)} \mid \mathcal{X}(R_i)). \tag{3}$$

To facilitate the presentation, we call the right-side part of '|' as the *condition* and the left-side part as the *outcome*.

4.2.3 Connection between the Two Problems. Both problems aim to predict the probability of an event where, given a path P exists, some other paths do not exist in the uncertain graph. The difference lies in *how paths are generated and the reference of 'other paths'*. In terms of the path generation, the Candidate-path-based Problem generates paths from s to t via Dijkstra+MC whereas the Random-walk-based Problem generates paths via random walks starting from s. In terms of the reference, 'other paths' in the Candidate-path-based Problem refer to the paths shorter than P in the candidate set CP whereas in the Random-walk-based Problem they refer to all paths in $\mathcal{S} \setminus \{P\}$.

The Candidate-path-based Problem is a special case of the Random-walk-based Problem since any instance (i.e., an event E_i) in the Candidate-path-based Problem can be mapped to an instance

(i.e., an event E'_i) in the Random-walk-based Problem. Specifically, given a candidate set $CP = \{P_1, \ldots, P_{|CP|}\}$, and a path $P \in CP$ the index of which is j ($1 \leq j \leq |CP|$), an event E_i ($1 \leq i < j$) can be mapped to an event E'_i where $\mathcal{S} = \{R_1 = P_1 \setminus P, \ldots, R_i = P_i \setminus P\}$.

Example 2. *Figure 3 shows a graph where the number assigned to each edge refers to the length and we do not show the edge probability for simplicity. Given the s-t pair, three candidate paths P, $P_1 = R_1 \cup \{(f,t)\}$ and $P_2 = R_2 \cup \{(e,f),(f,t)\}$, we have $Pr(E_2) = Pr(\overline{\mathcal{X}(P_1 \setminus P)} | \mathcal{X}(P_2 \setminus P))$. Suppose R_1 and R_2 are generated by random walks, then we have $\mathcal{S} = \{R_1, R_2\}$ and $Pr(E'_2) = Pr(\overline{\mathcal{X}(R_1)} | \mathcal{X}(R_2))$. Since $\overline{\mathcal{X}(P_1 \setminus P)} = \overline{\mathcal{X}(R_1)}$ and $\mathcal{X}(P_2 \setminus P) = R_2$, we have $E_2 \equiv E'_2$.*

Note that we only care about the existence probability of each path and how the path is generated will not impact the existence probability. Thus, the probability of having a random walk is irrelevant to the existence probability of the corresponding path generated by this walk. Furthermore, when generating a random walk, each out-going neighbor of the current node in the walk is treated equally regardless of the edge probability, which helps increase the diversity of $\mathcal{S}$, and the generality of the learning function.

The Learning Workflow. Figure 4 shows a high-level workflow of our learning process, and the mapping between each element in the Candidate-path-based Problem and the Random-walk-based Problem. Specifically, we first train a prediction function f by solving the Random-walk-based Problem and then transfer the trained function f to solve the Candidate-path-based Problem.

Transfer learning has been an important methodology to solve a learning problem where it is expensive or time-consuming to collect sufficient training data. Specifically, given a learning task in a source domain, transfer learning aims to improve the performance of the predictive function in the target domain by utilizing the knowledge in the source domain [60]. Our transfer learning strategy can be categorized into different classes based on different perspectives. In terms of the problem setting, it is inductive transfer learning [52] where we have the same source and target domains and different but related source and target tasks, since the Candidate-path-based Problem is a special case of the Random-walk-based Problem and both of them are defined in the same graph. In terms of the solution, it is parameter-based transfer learning [8, 41] where parameters or priors shared between the source and target domains are discovered, since we focus on transferring the learning function f.

In Section 4.3 and 4.4, we will introduce the training and testing procedure in detail respectively.

4.2.4 Benefits of studying the general problem. There are two major benefits of studying the Random-walk-based Problem instead of the Candidate-path-based Problem:

- Alleviation of the efficiency issue. The cost of generating a random-walk path in the Random-walk-based Problem can be much more controllable and smaller than that of generating a candidate path in the Candidate-path-based Problem. Given the predefined walk step ws and the average degree d in the uncertain graph, it only takes $O(ws \cdot d)$ to generate a random-walk path in the Random-walk-based Problem and the time complexity can be controlled by ws. On the other hand, generating a candidate path in the Candidate-path-based Problem incurs $O(|E| + |V| \log |V|)$ time which cannot be adjusted by hyperparameters.
- Alleviation of the generality issue. The Candidate-path-based Problem is a special case of the Random-walk-based Problem, which means that a prediction function to solve the Random-walk-based Problem can be directly transferred to solve the Candidate-path-based Problem. Furthermore, the prediction function trained in the Random-walk-based Problem should have more generalization power than the one trained in the Candidate-path-based Problem. Specifically, given an s-t pair in the Candidate-path-based Problem, candidate paths must be the paths between s and t; also, the training procedure may be too specific such that accurately solving the MPSP

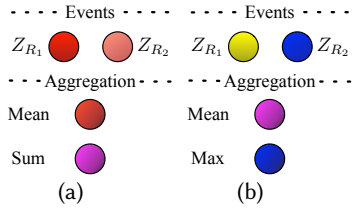


Fig. 5. Comparison among different choices of OP_{ot}.

problem for this s-t pair may not help solve this problem for a different query pair. In contrast, in the Random-walk-based Problem, a path is a random walk and can end at any possible node in the graph. Random-walk paths starting from the same node s create more diversity and flexibility than candidate paths between the same s-t pair, which can help obtain a more powerful and general prediction function.

4.3 Solving the Random-walk-based Problem

In this section, we will introduce our learning function f for predicting a value $\widetilde{Pr}(E'_i)$ to approximate $Pr(E'_i)$ in Equation 3. Specifically, we will first introduce how to represent edge and path embedding which are used to generate the embedding for the event E'_i with abstract element-wise embedding aggregators. Then we will discuss how to design these aggregators. Afterwards, we will introduce how the function f leverages the event embedding to predict $\widetilde{Pr}(E'_i)$.

Edge embedding. Given the embedding Z_v of size $\mathbb{R}^{1\times D}$ for each node v, we first generate the embedding $Z_{(u,v)}$ for each edge (u, v) based on the embeddings of u and v, i.e.,

$$Z_{(u,v)} = OP_{edge}(Z_u, Z_v),$$

where $OP_{edge} : 2 \times \mathbb{R}^{1\times D} \rightarrow \mathbb{R}^{1\times D}$ refers to an embedding aggregator. Note that we assume that node embedding is provided in advance and will not be updated during training.

Path embedding. Given a random-walk path $R = \{e_1, e_2, \ldots, e_{|R|}\}$, the embedding Z_R of the path R is generated based on the embeddings of all edges in this path, i.e.,

$$Z_R = OP_{path}(Z_{e_1}, \ldots, Z_{e_{|R|}}),$$

where $OP_{path} : |R| \times \mathbb{R}^{1\times D} \rightarrow \mathbb{R}^{1\times D}$ refers to an aggregator.

Event embedding. Given an event E'_i with the corresponding set $\mathcal{S}$ of random-walk paths $R_1, R_2, \ldots, R_{|\mathcal{S}|}$, the embedding of E'_i is generated based on the embeddings of all paths in $\mathcal{S}$, i.e.,

$$Z_{E'_i} = [OP_{ot}(Z_{R_1}, \ldots, Z_{R_{i-1}}, Z_{R_{i+1}}, \ldots, Z_{R_{|\mathcal{S}|}})||Z_{R_i}],$$

where $OP_{ot} : (|\mathcal{S}|-1)\times\mathbb{R}^{1\times D} \rightarrow \mathbb{R}^{1\times D}$ is an aggregator, $[\cdot||\cdot]$ refers to the embedding concatenation operator, Z_{R_i} at the right side of $[\cdot||\cdot]$ refers to the embedding of the condition, and the left-side part refers to the embedding for the outcome in Equation 3.

Choice of OP_{edge}, OP_{path} and OP_{ot}. For OP_{edge}, we have tried multiple popular aggregators (e.g, Mean, Hadamard, Weighted L1 and Weighted L2) used in existing literature [9, 20, 28, 33, 51]. We find that the choice barely impacts the prediction accuracy. Thus, we randomly choose the Weighted L1 aggregator (i.e., $|Z_u - Z_v|$). For OP_{path}, we consider commonly used aggregators for subgraph embedding since a path can be treated as a subgraph. Such aggregators include Sum, Mean and Max-pooling [4, 14, 21, 57]. We adopt the Sum aggregator since it has better expressive power to make the aggregation results, which may consist of similar features, distinguishable [57].

For OP_{ot}, we adopt the Mean aggregator since it has more expressive power than the Max-pooling aggregator and is more capable of reducing the impact brought by similar or repeatitive events than the Sum aggregator, as shown below.

Example 3. *Figure 5 compares different choices of OP_{ot} in two cases (a) and (b) where the similarity of R_1 and R_2 is reflected by their colors. In case (a), we are interested in computing $Pr(\overline{\mathcal{X}(R_1)}, \overline{\mathcal{X}(R_2)}|\mathcal{X}(R_x))$ conditioned on some R_x. Since R_1 and R_2 are quite similar, we have $Pr(\overline{\mathcal{X}(R_1)}, \overline{\mathcal{X}(R_2)}|\mathcal{X}(R_x)) \approx Pr(\overline{\mathcal{X}(R_1)}|\mathcal{X}(R_x))$. Thus, $OP_{ot}(Z_{R_1}, Z_{R_2})$ is expected to be similar to Z_{R_1} or Z_{R_2}, which may not be achieved by the Sum aggregator. In case (b), R_1 and R_2 are very different. Thus, $OP_{ot}(Z_{R_1}, Z_{R_2})$ should combine features/information in both Z_{R_1} and Z_{R_2} and is expected to be different from Z_{R_1} and Z_{R_2}, which may not be achieved by the Max-pooling aggregator.*

Learning function f. We adopt a simple Multi-Layer Perceptron (MLP) Neural Network as the learning function f which only consists of three layers (i.e., the input layer, one hidden layer and the output layer) and is shown to be highly efficient and effective in our experiments. Specifically, given the event embedding $Z_{E'_i}$, f is defined as below:

$$f(Z_{E'_i}) = \text{ReLU}\left(\text{BN}(Z_{E'_i} \times W_1)\right) \times W_2 = \widetilde{Pr}(E'_i)$$

where ReLU stands for the Rectified Linear Unit activation function [24, 25], BN stands for the batch normalization technique [35] used in mini-batch stochastic gradient descent, $W_1 \in \mathbb{R}^{(2\times D)\times D}$ and $W_2 \in \mathbb{R}^{D\times 1}$ refer to trainable parameters.

Loss function $\mathcal{L}$. Suppose we have a set O of training pairs and each training pair $(\widetilde{Pr}(E'), Pr(E'))$ consists of the predicted probability and the ground-truth probability of an event E'. We aim to minimize the mean absolute error over all training pairs, i.e.,

$$\mathcal{L} = \frac{\sum_{(\widetilde{Pr}(E'), Pr(E'))\in O} |\widetilde{Pr}(E') - Pr(E')|}{|O|}.$$

Note that since the exact $Pr(E')$ is hard to obtain, we use a number (i.e., 100) of Monte-Carlo simulations to compute an approximate value as the ground-truth, which ensures the efficiency of training pairs generation and is sufficient for the learning-based method to produce high-quality results for the MPSP problem.

The training process. Algorithm 7 describes the training process. Given a sampling ratio sr, we first sample $sr \cdot |V|$ starting nodes. For each starting node, we generate a set $\mathcal{S}$ of rw random-walk paths with steps ws. Then, we use path embedding to prepare each training pair $(Pr(E'_i), Z_{E'_i})$. Afterwards, we use each training pair to predict $\widetilde{Pr}(E'_i)$ and minimize the loss function.

4.4 Transfer Learning to Solve the Candidate-path-based Problem

We directly transfer the trained prediction function f to solve the MPSP problem, as shown in Figure 4. Algorithm 8 shows the testing procedure which uses the learning function f trained in Algorithm 7 for the Learning-based Probability Approximation (LPA) in Phase 2. In each round (i.e., Line 3), we compute the probability of each path in CP being the shortest path based on the predicted $\widetilde{p}$. There are at most $O(m)$ inner loops to update and compute $\widetilde{p}$ (Lines 6 to 15). Lines 9 to 11 take $O(|CP[-1]| \cdot D)$ time where $|CP[-1]|$ refers to the length of the longest/last path in CP. Line 13 dominates the cost from Lines 12 to 15 and it takes $O(D^2)$ time. Thus, each round takes $O(m \cdot D \cdot (|CP[-1]| + D))$, and the total time complexity is $O(m^2 \cdot D \cdot (|CP[-1]| + D))$. Compared to the non-learning based optimization of Phase 2 (i.e., Algorithm 4) for each round (i.e., $O(m \cdot D \cdot (|CP[-1]| + D))$ vs. $O(N \cdot \sum_{P\in CP} |P|)$), the learning-based method incurs notably lower

Algorithm 7: The practical training procedure

Input : An uncertain graph $\mathcal{G} = (V, E, l, p)$, the sampling ratio sr, the number rw of random walks, the random walk step ws, the batch size b and training parameters W_1 and W_2.

$Pairs \leftarrow \emptyset$;
for $iter = 1$ *to* $sr \cdot |V|$ **do**
  Sample a starting node s;
  $\mathcal{S} \leftarrow$ a set of random-walk paths $R_1, R_2, \ldots, R_{rw}$, each has ws edges and starts from s;
  $Z_{all} \leftarrow \sum_{i=1}^{rw} Z_{R_j}$;
  for $i = 1$ *to* rw **do**
    Compute $Pr(E'_i)$;
    $Z_{E'_i} \leftarrow [\frac{Z_{all} - Z_{R_i}}{rw-1} || Z_{R_i}]$;
    Add the pair $(Pr(E'_i), Z_{E'_i})$ into $Pairs$;
while *not converged* **do**
  Shuffle $Pairs$;
  foreach *batch of size b in Pairs* **do**
    $loss \leftarrow 0$;
    foreach $(Pr(E'_i), Z_{E'_i})$ *in the batch* **do**
      $\widetilde{Pr}(E'_i) \leftarrow f(Z_{E'_i}) = Relu\left(BN(Z_{E'_i} \times W_1)\right) \times W_2$;
      $loss \leftarrow loss + |\widetilde{Pr}(E'_i) - Pr(E'_i)|$;
    Minimize $loss/b$;

Algorithm 8: Phase 1 + Learning-based Probability Approximation (LPA)

Input : An uncertain graph $\mathcal{G} = (V, E, l, p)$, an integer m, the source and target nodes s and t, the embedding size D, and trained parameters W_1 and W_2.

Output : An approximate MPSP(s, t).

Get the candidate set CP from Phase 1 in Algorithm 1;
Order all paths in CP in increasing order of length;
for $j = 1$ *to* $|CP|$ **do**
  $Z_{ot} \leftarrow$ a vector of D 0s;
  $P \leftarrow CP[j], \widetilde{p} \leftarrow 0$;
  for $i = 1$ *to* $j - 1$ **do**
    $Pr(\mathcal{X}(P_i \setminus P)) \leftarrow 1$;
    $Z_{P_i \setminus P} \leftarrow$ a vector of D 0s;
    foreach *edge* $(u, v) \in P_i \setminus P$ **do**
      $Pr(\mathcal{X}(P_i \setminus P)) \leftarrow Pr(\mathcal{X}(P_i \setminus P)) \times p((u, v))$;
      $Z_{P_i \setminus P} \leftarrow Z_{P_i \setminus P} + Z_{(u,v)}$;
    $Z_{E_i} \leftarrow [Z_{ot}/(i-1) || Z_{P_i \setminus P}]$;
    $\widetilde{Pr}(E_i) \leftarrow f(Z_{E_i}) = Relu\left(BN(Z_{E_i} \times W_1)\right) \times W_2$;
    $Z_{ot} \leftarrow Z_{ot} + Z_{P_i \setminus P}$;
    $\widetilde{p} \leftarrow \widetilde{p} + Pr(\mathcal{X}(P_i \setminus P)) \cdot \widetilde{Pr}(E_i)$;
  $\widetilde{Pr}(SP_s^t(P)) \leftarrow (1 - \widetilde{p}) \times Pr(\mathcal{X}(P))$;
return $\arg\max_{P \in CP} \widetilde{Pr}(SP_s^t(P))$

time complexity since $m \cdot D$ is quite comparable to N (e.g., $m = 20$, $D = 128$ and $N = 1000$) but $\sum_{P \in CP} |P|$ can be notably larger than $(|CP[-1]| + D)$ in practice.

Table 1. Statistics of datasets

Dataset	|V|	|E|
Beijing (BJ)	75,958	109,741
New York (NY)	264,346	365,050
Chengdu (CD)	631,213	1,077,967
Northeast (NE)	1,524,453	3,868,020
California (CA)	1,965,206	2,766,607

5 EXPERIMENT

Experiment Purposes. In this section, we will conduct extensive experiments to demonstrate the efficiency and effectiveness of our proposed methods. We aim to show that (1) our learning-based method LPA is very effective and competitive with our non-learning based method FPA under different evaluation metrics (Exp 1 and Exp 2), (2) optimizing the efficiency of Phase 2 is very important in boosting the whole workflow (Exp 3), (3) FPA and LPA are very efficient and notably outperform PA (Exp 4), (4) the training process of LPA is highly scalable (Exp 5), (5) hyperparameters have impact on the performance of LPA (Exp 6), and (6) LPA has strong generalization power such that it can work effectively in different problem settings from the one it was trained (Exp 7).

5.1 Experimental Setup

Datasets. To tackle the hard instances of this problem and thus better compare methods' performance, we focus on experiments on road networks which have significantly larger graph sizes than networks of other types in discovered applied domains (e.g., sensor networks and brain networks). Such experiments are also very useful in reality since the studied problem MPSP has strong applications in recommendation of the shortest routes in road networks where edges are uncertain due to unexpected traffic jams or blockage [10, 11, 31, 58]. The five real-world road networks we use are shown in Table 1. Beijing (BJ), Chengdu (CD) and New York (NY) are from [48], and Northeast (NE) and California (CA) are available at [12]. For experiments on other domains, please refer to Exp 8.

Top-k MPSPs. We follow [53] to compare the returned top-k paths with the highest probabilities by varying $k \in \{1, 5, 10\}$ with 1 as the default value. If the number of candidate paths is smaller than or equal to k, all candidate paths will be returned.

Methods for Comparison. We compare our proposed learning-based method LPA (Algorithm 8) against the following baselines. Since all methods share the same Phase 1 (i.e., Lines 2 to 4 in Algorithm 1) and only differ in the strategy for choosing candidate paths in Phase 2, for simplicity, we only use their strategy name in Phase 2 to represent the whole process including Phase 1.

- PA (Algorithm 3) [53]: the original implementation of the Luby-Karp Algorithm [38] which computes the probability of each candidate path being the shortest path. We optimize this method by only sampling edges from candidate paths in Phase 2.
- FPA (Algorithm 4): our proposed method with a fast probability estimation technique in Phase 2. Since PA and FPA produce equivalent results, the performance of PA will only be reported in the efficiency comparison.
- Random: a method which chooses k paths uniformly at random from candidate paths found in Phase 1 as the result. We report its average performance over five independent runs.
- Reverse Ranking (RR): a method which returns the top-k paths with the lowest probabilities computed by FPA. This method produces the *worst possible solution* based on the candidate paths generated in Phase 1. The performance gap between RR and the best method FPA can be used to help judge a method's performance. Specifically, suppose we have a method which produces

similar results as FPA. If RR and FPA also produce similar results, we cannot really tell whether this method is effective, since any method (i.e., Random and LPA) can easily produce competitive results with FPA. On the other hand, if the performance gap between RR and FPA is large, it is more obvious to tell that this method is more effective than in the previous case.

Parameter Setting. Following [53], we set $m = 20$ in Phase 1 and the number of simulations $N = 1000$ in Phase 2 for both PA and FPA. For our learning-based method LPA, we use Node2Vec [28] with default settings where the embedding size $D = 128$ to train the node embedding in advance; the node embedding will not be updated during the training process. To generate the training data for the Random-walk-based Problem, we randomly select a number (i.e., $sr \times |V|$) of nodes and, for each of these nodes as a starting node, we generate a set of rw random walks with the walk step ws. To show the robustness and effectiveness of our method, we do not perform the costly grid search over all parameters' choices to find the best combination. Instead, we only perform a grid search over the sampling ratio sr while setting the other parameters as default values. Specifically, the number rw of random walks per starting node is set to 20 and the walk step ws is set to 30 by default. These default values are neither too small nor too large such that we can produce good results without overfitting. We perform a grid search over $sr \in \{0.0005, 0.001, 0.01, 0.1, 0.2\}$ with a validation set of 100 randomly sampled s-t pairs. We choose the ratio, which leads to the best prediction performance or is the smallest one with the converged performance, as the default value. The default sr is set as 0.1, 0.2, 0.001, 0.1, 0.1 in BJ, NY, CD, NE and CA respectively.

Test Cases. For each dataset, we randomly sample 500 $s-t$ pairs as the test cases for the Candidate-path-based Problem. Notably, this test size is five times larger than the one in PA [53] and can better reflect all methods' performance.

Edge Probability Setting. To consider hard instances of this problem, candidate paths for each s-t pair should be sufficient and diverse. Thus, we need to assign each edge with a high probability value to avoid probability diminishing for discovering candidate paths for each s-t pair where s can be many hops away from t. By default, we randomly assign a probability to each edge from the continuous range $[0.90, 1)$. In later generalization test (Exp 7), we will conduct experiments with more ranges.

Evaluation Metrics. To evaluate the effectiveness, we adopt two evaluation metrics. The first one is the ***Average of the Average Probability*** **(AAP)**. It refers to the average of the average probability of returned top-k paths for every test s-t pair. However, we think that this metric cannot well reflect a method's performance when the distribution of the average probability of returned top-k paths is skewed. For example, suppose we have ten s-t pairs, the best average probabilities for nine of these ten s-t pairs and the remaining pair are 10^{-5} and 1 respectively, and the worst possible average probability for the nine and the remaining pair is infinitely small (say 0). The AAP of a method, which finds the optimal solution for the remaining pair but finds the worst solution for the other nine pairs, will be 0.1. This AAP is greater than 9×10^{-6} achieved by a method which finds the best solution for the nine pairs but finds the worst solution for the remaining pair. If all test pairs are equally important, which is the case in this work and [53], the latter method with a smaller AAP is actually a better one because it produces the best solution for most test pairs.

To alleviate the above issue of the AAP metric, we propose another evaluation metric called ***Aggregated Rank (AR)***. Given the top-k paths returned by a method for a test pair, the AR of these paths is the sum of their ranks based on their probabilities computed by FPA in decreasing order. For example, the path whose probability computed by FPA is the highest will have rank 1. Thus, if the AR achieved by a method is smaller, the performance is better and closer to FPA's performance. This metric focuses on the quality of path ranking instead of the actual probability values, and thus is more robust to the skewed distribution of the average probability of the returned top-k paths.

Table 2. Comparisons of the AARs achieved at different percentiles. When $k = 1, 5$ and 10, the AARs achieved by FPA are 1, 15 and 55 and the AARs achieved by RR are 20, 90 and 155 respectively. Note that the AARs achieved by FPA and RR are consistent across datasets and thus not reported in the table below for simplicity.

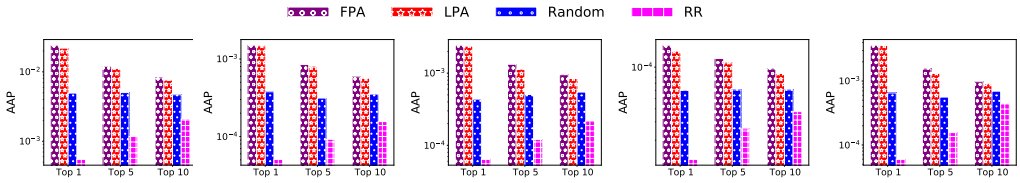
Dataset	k	Method	0th	25th	50th	75th	80th	90th	95th	100th	Mean
BJ	1	Random	1.4	8.6	10.4	12.4	12.8	14	14.8	17.8	10.36
		LPA	1	1	1	1	1	2	2	14	1.26
	5	Random	22.8	49	52.1	55.75	56.8	59	61.11	69	52.25
		LPA	15	15	15	17.75	18	21	27	43	17.12
	10	Random	55	99.8	104	108.8	110.08	112.8	114.24	123	103.99
		LPA	55	55	55	59	61.4	68.2	75	101	58.77
NY	1	Random	3	8.6	10.4	12.2	12.6	13.8	14.71	17	10.44
		LPA	1	1	1	1	1	1	1	6	1.02
	5	Random	18	48.6	52.8	56.4	57.44	59.4	61	68.6	52.63
		LPA	15	15	15	15	15	15	16	41	15.34
	10	Random	76.8	100.6	104.6	108.6	109.2	111.2	113	117.8	104.19
		LPA	55	55	55	55	55	55	56	88	55.47
CD	1	Random	3.8	9	10.6	12.4	13	13.8	14.51	17.8	10.59
		LPA	1	1	1	1	1	1	1	9	1.05
	5	Random	30.6	48.8	52.2	55.6	56.6	58.62	60.31	70.6	52.16
		LPA	15	15	15	15	15	16	18	49	15.68
	10	Random	67.2	101.25	105.2	109	110	114	115.8	120.8	105.26
		LPA	55	55	55	55	55	56	60.55	108	56.19
NE	1	Random	3.4	8.6	10.6	12.2	12.6	14	14.6	17.6	10.37
		LPA	1	1	1	1	1	1	1	6	1.05
	5	Random	22	48.2	52	55.8	56.4	59.2	61.02	68.4	51.60
		LPA	15	15	15	15	15	15	18	49	15.52
	10	Random	58.4	99.6	104.2	108.2	108.8	111.8	114.2	119.6	102.81
		LPA	55	55	55	55	55	56	62.65	121	56.14
CA	1	Random	3.2	8.6	10.3	12.35	12.8	13.8	14.71	18.2	10.34
		LPA	1	1	1	1	1	1	2	8	1.12
	5	Random	27.8	49	52.6	56.4	57.4	59.4	61.4	67.2	52.60
		LPA	15	15	15	17	19	24	29	53	17.53
	10	Random	55	100.4	104.7	108.6	109.4	111.8	113.51	119	104.23
		LPA	55	55	55	62	69	82	90.55	123	61.63

Since there are more than one test s-t pair, we report ***Average Aggregated Rank (AAR)*** which denotes the average of AR of returned top-k paths for every test pair.

Environments. We conduct all experiments on a Linux server with Intel Xeon E5 (2.60 GHz) CPUs and 512 GB RAM. The implementation can be found in [1].

5.2 Experimental Results

Exp 1 - AAR Comparison. Table 2 shows the AARs achieved by all methods at different percentiles where the performance of FPA and RR is reported in the caption. Our proposed learning-based method LPA produces very competitive results (i.e., exactly the same result in most cases) with FPA, and significantly outperforms other methods. Specifically, we have two key observations: (i) When $k = 1$, the 80th percentile is 1 and the 95th percentile is at most 2 across all datasets, which indicates that LPA produces the same and nearly the same result as FPA in 80% and 95% of test cases respectively; (ii) When $k = 5$ and $k = 10$, the 50th percentile is 1 across all datasets, and, more



(a) BJ (b) NY (c) CD (d) NE (e) CA

Fig. 6. Comparison of AAP achieved by different methods

Table 3. Running time (s) cost decomposition of PA

	BJ	NY	CD	NE	CA
Phase 1	7.4	22.6	68.3	118.6	186.2
Phase 2	7.9	32.1	18.3	55.7	16.6

Table 4. The average number of edges in candidate paths

	BJ	NY	CD	NE	CA
Length	137	460	375	942	370

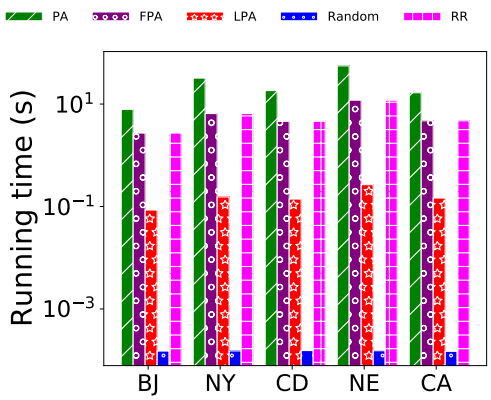


Fig. 7. Average time cost in Phase 2

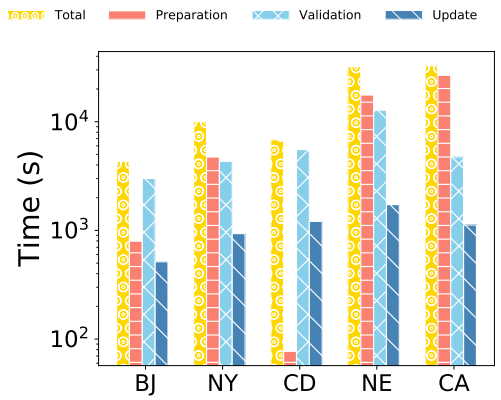


Fig. 8. Training time decomposition of LPA

notably, LPA produces the same and nearly the same result as FPA in 80% and 90% of test cases respectively in NY, CD and NE. These observations indicate the high effectiveness of our method.

Exp 2 - AAP Comparison. Figure 6 shows the AAP achieved by all methods across datasets. Again, our proposed learning-based method LPA is very competitive with FPA and notably outperforms the other methods. We have the following key observations: (i) For LPA, in the worst case where $k = 5$ in CD, its AAP is 85.5% of FPA's AAP. In the best case where $k = 1$ in CA, its AAP is 99.6% of FPA's AAP; (ii) For Random, in the worst case where $k = 1$ in CA and CD, its AAP is only around 18% of LPA's AAP. In the best case where $k = 10$ in CA, its AAP is only around 76% of LPA's AAP; (iii) For RR, in the worst case where $k = 1$ in CD and NY, its AAP is only around 3% of LPA's AAP. In the best case where $k = 10$ in CA, its AAP is only 48% of LPA's AAP; (iv) The AAP achieved by RR increases as k increases, since it returns k paths with the lowest probabilities; (v) The performance gap between Random and FPA becomes smaller as k increases since the probability of a path being chosen by both Random and FPA increases as k. Since $m = 20$, the probability that the Random method chooses correct paths can be very high especially when $k = 10 = 1/2m$. Imagine when k is equal to the number of candidate paths, the results produced by Random and FPA will be the same; (vi) When $k = 1$, the performance gap w.r.t. AAP between FPA and LPA is more notable in NE than that in CA but the performance of LPA w.r.t. AAR is better in NE as shown in Table 2, which indicates that we should not solely rely on AAP.

Table 5. LPA's training time compared with PA's average Phase 2 cost for test queries and the total Phase 2 cost for all possible queries

	BJ	NY	CD	NE	CA
Training time/ Avg. Phase 2 cost	548	312	372	575	1963
Training time/ Total Phase 2 cost	1.9E-7	8.9E-9	1.9E-9	4.9E-10	1.0E-9

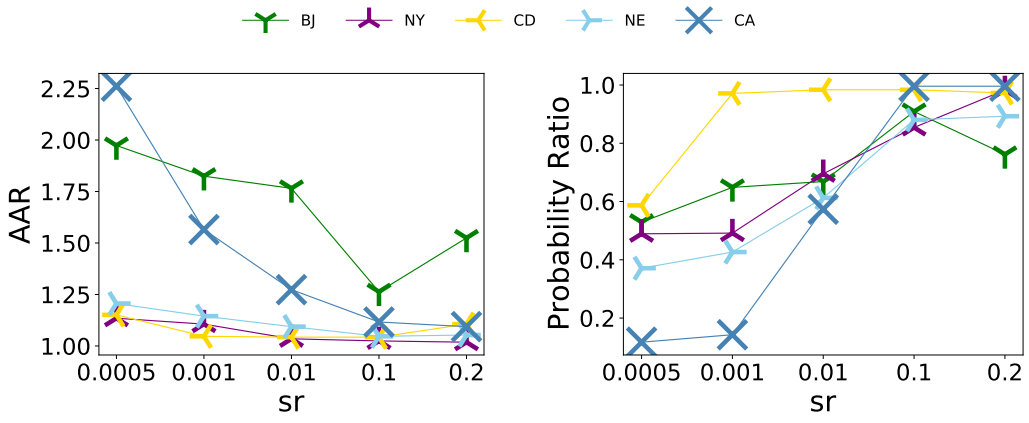
Table 6. Impact of the number m of iterations in Phase 1

Data	k=1		k=5		k=10	
-set	m=20	m=30	m=20	m=30	m=20	m=30
BJ	2.41E-2	2.41E-2	1.20E-2	1.21E-2	8.17E-3	8.34E-3
NY	1.50E-3	1.50E-3	8.48E-4	8.48E-4	5.89E-4	5.97E-4
CD	2.44E-3	2.44E-3	1.33E-3	1.33E-3	9.52E-4	9.64E-4
NE	1.62E-4	1.62E-4	1.21E-4	1.21E-4	9.63E-5	9.76E-5
CA	3.63E-3	3.63E-3	1.55E-3	1.55E-3	9.76E-4	9.91E-4

Exp 3 - Running Time Decomposition of PA. Table 3 shows the decomposition of the average running time of PA over all test pairs. The Phase 2 accounts for a notable portion of the total running time and it can easily dominate the total cost. That is because Phase 1 can be trivially extended to handle the case for multiple queries with the same source node but different target nodes. Specifically, we can make the procedure of Dijkstra+MC continue until all edges are sampled or no new nodes can be reached, and execute Phase 2 separately for each individual query. Similarly, we can also handle the case for queries with the same target node but different source nodes by conducting the aforementioned procedure on the graph where all edge directions are reversed. These two cases are very useful for answering multiple queries and in these two cases the cost of Phase 2 can account for a significant portion of the total cost and the Phase 1 cost can be very similar to the counterpart in a single query pair. We also find that the graph size which directly impacts Phase 1 cost does not necessarily have a notable impact on Phase 2 cost, and we think that, as shown in Table 4, Phase 2 cost has positive correlations with the average number of edges in candidate paths generated in test cases.

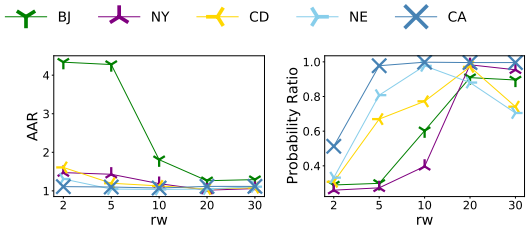
Exp 4 - Time Cost Comparison of Phase 2. Since all methods share the same process in Phase 1, we focus on comparing the running time of their Phase 2 whose cost can easily dominate the whole process as illustrated in Exp 3. Figure 7 shows the average Phase 2 cost over all test cases and we can see that FPA is 3x - 5x faster than PA, which demonstrates the efficiency of our fast sampling estimation technique. Moreover, our learning method LPA can achieve 92x - 210x speedups over PA, which is a significant improvement and demonstrates the superiority of our light-weight learning algorithm. Even though Random is very fast, it is not comparable to others since its effectiveness is quite limited.

Exp 5 - Training Time Decomposition. Figure 8 shows the training time decomposition of LPA on different datasets where 'Preparation' refers to the process of random walk generation and the ground-truth probability computation, 'Validation' refers to validating the model performance with the aforementioned 100 validation s-t pairs and 'Update' refers to the back propagation process where model trainable parameters are updated. The 'Preparation' and 'Validation' procedures together dominate the total cost whereas the 'Update' cost only accounts for 3%-20% of the total cost. The 'Preparation' cost heavily depends on the default sampling ratio sr which is set as 0.1, 0.2, 0.001, 0.1, 0.1 in BJ, NY, CD, NE and CA respectively. To have a better understanding about the



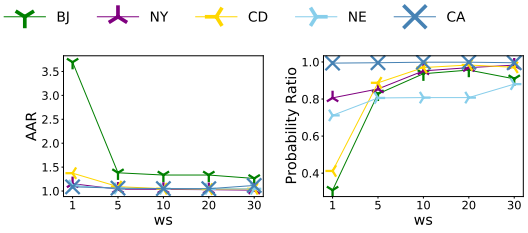
(a) (b)

Fig. 9. Impact of the sampling ratio



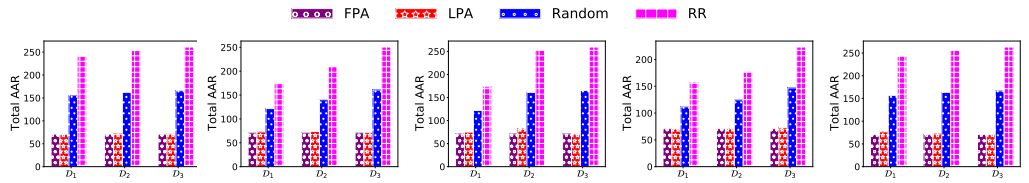
(a) (b)

Fig. 10. Impact of the number of random walks



(a) (b)

Fig. 11. Impact of the random walk step

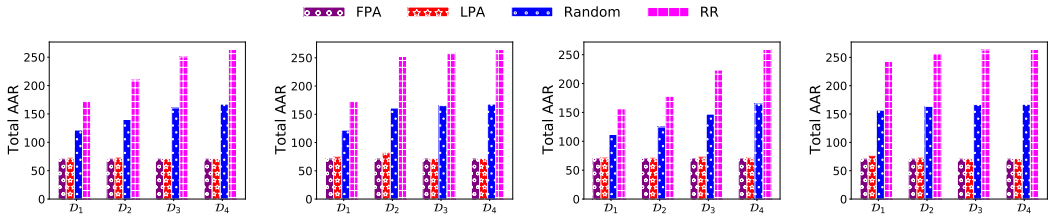


(a) BJ (b) NY (c) CD (d) NE (e) CA

Fig. 12. Generalization Tests of LPA trained in the same dataset with the distribution $\mathcal{D}_4$ but deployed in other distributions

training efficiency, we compare the training process with Phase 2 of PA, as shown in Table 5. In the worst case, the ratio of the training time against the average Phase 2 cost of PA over all test cases is only 1963. It means that, in the worst case, the training time is equal to the total cost of Phase 2 for 1963 s-t pairs, which is pretty small compared to the number of all possible testing s-t pairs. To see this, in Table 5, we also show the ratio of the training time against the total Phase 2 cost of all possible pairs (approximated by the average Phase 2 cost of PA over all test cases times the number of possible pairs in the graph). The ratio is at most 1.9E-7, which indicates that the training process is highly efficient and deployable in practice.

Exp 6 - Parameter Study. Table 6 shows the impact of m on the performance of FPA. For paths returned by $m = 20$ and $m = 30$, they are identical for $k = 1$ and $k = 5$, and very similar for



(a) NY (b) CD (c) NE (d) CA

Fig. 13. Generalization Tests of LPA trained in BJ with the distribution $\mathcal{D}_4$ but deployed in different datasets

$k = 10$. We also conduct experiments for larger m which is not included in the paper since $m = 20$ is the convergence point. Figure 9, Figure 10 and Figure 11 show the impact of three parameters, namely the sampling ratio, the number of random walk per starting node and the random walk step, on the AAR and Probability Ratio (i.e., the ratio of the probability of the top-1 path returned by LPA over the one of the top-1 path returned by FPA) respectively when $k = 1$. We have two key observations: (i) When the value of any one parameter increases before reaching the convergence point, the performance (in terms of both AAR and Probability Ratio) increases and may degrade after the convergence point probably because of the overfitting issue; (ii) LPA can achieve small AARs even with small parameter values (i.e., coarse-grained information), which indicates the high effectiveness of LPA; (iii) When LPA achieves a relatively low AAR with small parameter values, there is still considerable improvement space for the Probability Ratio. This observation may be caused by the reason that the candidate paths with very similar ranks may have notably different probabilities such that the Probability Ratio can be quite low even with a small AAR. Thus, LPA needs more information with larger parameter values to give a more precise path ranking.

Exp 7 - Generalization Test. To test the generalization and robustness of LPA, we report the performance of LPA in different problem settings from the setting it was trained. To construct different settings, we use four independent distributions $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$ and $\mathcal{D}_4$ by uniformly sampling a probability for each edge from the ranges $[0.6, 1)$, $[0.7, 1)$, $[0.8, 1)$ and $[0.9, 1)$ respectively, where $\mathcal{D}_4$ is the default distribution used in previous experiments.

Figure 12 shows the Total AAR (i.e., the sum of the AARs of the top-1, top-5 and top-10 paths returned) of LPA which is trained in the same dataset with distribution $\mathcal{D}_4$ but deployed in different distributions. The generalization performance is very promising since LPA can achieve nearly identical Total AAR to the one (i.e., 71) achieved by FPA in most cases. One possible reason is that instances tend to be harder under $\mathcal{D}_4$ than the ones under other distributions since candidates paths under $\mathcal{D}_4$ can be more diverse, and thus LPA trained under $\mathcal{D}_4$ can effectively solve instances under other distributions. It is worth noting that the performance of Random and RR improves as the sampling range increases. The reason is that, with a larger sampling range, the number of generated candidate paths tends to be smaller such that the total AAR of Random and RR will correspondingly become smaller.

Figure 13 shows the performance of LPA trained in BJ with $\mathcal{D}_4$ but deployed in the other datasets with different distributions. LPA under this setting can produce very competitive results with FPA and LPA under the previous setting (i.e., trained and deployed in the same dataset) by comparing with Figure 12. There are two possible reasons. First, our prediction target is the path existence probability which treats each path as a set of edges and does not need to know the topological information (e.g., how edges are connected) which is already contained in the pre-trained node embedding. Such design greatly improves the generalization power of LPA. Second, test cases tend to be harder and information tends to be richer in BJ than the ones in other datasets, since LPA needs larger parameters in BJ to converge (as shown in Figures 9 - 11).

Table 7. Comparisons of achieved AARs at different percentiles when $k = 1$ in Brain and Intel.

Dataset	Method	0th	25th	50th	75th	80th	90th	95th	100th	Mean
Brain	RR	1	1	2	2	2	3	3	4	1.87
	Random	1	1	1.4	1.6	1.8	2.0	2.2	3.2	1.45
	LPA	1	1	1	1	1	1	1	1	1
	FPA	1	1	1	1	1	1	1	1	1
Intel	RR	1	2	2	2	2	3	3	3	1.93
	Random	1	1.2	1.4	1.6	1.8	2.0	2.2	2.8	1.46
	LPA	1	1	1	1	1	1	1	1	1
	FPA	1	1	1	1	1	1	1	1	1

Table 8. Comparisons of AAP in Brain and Intel.

Dataset	FPA	LPA	Random	RR
Brain	0.942	0.942	0.593	0.309
Intel	0.936	0.936	0.573	0.259

Table 9. Running time comparison in Brain and Intel.

Dataset	PA	FPA	LPA	Random	RR
Brain	3.2E-3	3.1E-3	8.2E-5	3.1E-5	3.1E-3
Intel	3.5E-3	3.4E-3	8.5E-5	3.1E-5	3.4E-3

Exp 8 - Experiments on Other Domains. To make the experiment more complete, we conduct experiments on networks from other domains. These networks inherently have notably smaller sizes than road networks, and were also used by the state-of-the-art [53]. Specifically, we consider two networks, one sensor network (Intel) [45] with 54 nodes and 1289 edges and one brain network (Brain) [13] with 116 nodes and 6670 edges. All experiments are conducted with aforementioned default parameter settings. Due to the simple topology and small-scale graph size, the generated candidate paths are quite limited. Thus, we only conduct experiments with $k = 1$. Table 7 and Table 8 compare the performance of different methods in terms of AARs and AAP respectively. Our learning-based method LPA achieves exactly the same performance as the gold standard FPA and notably outperforms other methods. Table 9 compares the running time. Since the average lengths of candidate paths are pretty small in these two datasets, the superiority of FPA over PA is not obvious. However, LPA can still notably outperform PA and FPA since it does not involve costly simulations for probability approximation.

6 RELATED WORK

Shortest-paths-related problems. There have been some shortest-paths-related problems in uncertain graphs studied in the literature. The problem of finding threshold-based shortest-path queries in uncertain graphs is investigated in [10, 11, 58] where [10, 11] consider the appearance of an edge to be dependent on other edges. The work [61] solves the MPSP problem via a filtering-and-verification framework which enumerates a number of candidate paths between the source and destination nodes in increasing order of length. Afterwards, a sampling approach (i.e., the Luby-Karp algorithm [38]) is used to approximate the probability of each candidate path being the shortest path. This strategy [61] suffers from the scalability issue and may have limited effectiveness since a large number of paths will be enumerated solely based on length without considering edges' probabilities such that the correct path may not be enumerated. To alleviate these issues, Monte Carlo simulations with Dijkstra's algorithm (Dijkstra+MC) are combined in [53] to efficiently generate candidate paths. In Dijkstra+MC, the outgoing edges of the node, currently visited by the Dijkstra algorithm, will be sampled based on their probabilities and the decision upon the next move is only based on sampled edges. It has been proven that, with a high probability, the correct path will be returned as the solution from the candidate path set with a small number of Dijkstra+MC runs. Despite the significant progress being made, there is still considerable space for efficiency improvement for this problem. Specifically, given the generated candidate paths, a large

number of simulations is required by the Luby-Karp algorithm to approximate the probability of each candidate path being the shortest path, which can easily dominate the total time cost. In this paper, we propose a machine-learning method to compute probabilities without costly simulations.

Transfer learning. In terms of the problem setting, transfer learning can be categorized into three classes. The first one is transductive transfer learning [17, 54, 59] where we have the same source and target tasks but different source and target domains. The second one is unsupervised transfer learning [16, 56] where the source and target domains, and the source and target unsupervised tasks are different but related. Our method falls into the third class, namely inductive transfer learning [52]. In terms of the solution, transfer learning can be categorized into four classes. The first one is instance-based transfer learning [15, 59] which re-weights some labeled data in the source domain for utilization in the target domain. The second one is feature-based transfer learning [42, 52]. It aims to find good feature representations to reduce difference between the source and target domains. The third one is relational-knowledge-based transfer learning [18, 46, 47] which aims to build mapping of knowledge between the relational source and target domains. Our method falls into the fourth class, namely parameter-based transfer learning [8, 41].

7 CONCLUSION

In this paper, we study the problem of finding the most probable shortest paths in uncertain networks. The state-of-the-art adopts a two-phase approach where Phase 1 generates candidate paths and Phase 2 estimates the probabilities of each candidate path and return the one with the highest probability. The original Phase 2 requires a large number of costly simulations and can easily dominate the total time cost. In this paper, we focus on optimizing Phase 2 by proposing a non-learning based and a learning-based approach. The non-learning based method significantly reduces the sample size in each simulation and the learning-based method with transfer learning directly computes the probability of each candidate path without calling costly simulations. Extensive experiments demonstrate the efficiency and effectiveness of our methods.

ACKNOWLEDGMENT

Zhifeng Bao is in part supported by ARC Discovery Project DP220101434.

REFERENCES

[1] 2023. https://github.com/rmitbggroup/MPSP.

[2] Eytan Adar and Christopher Re. 2007. Managing uncertainty in social networks. *IEEE Data Eng. Bull.* 30, 2 (2007), 15–22.

[3] Saurabh Asthana, Oliver D King, Francis D Gibbons, and Frederick P Roth. 2004. Predicting protein complex membership using probabilistic network reliability. *Genome research* 14, 6 (2004), 1170–1175.

[4] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. 2019. Simgnn: A neural network approach to fast graph similarity computation. In *WSDM*. 384–392.

[5] Michael O Ball. 1986. Computational complexity of network reliability analysis: An overview. *IEEE Transactions on Reliability* 35, 3 (1986), 230–239.

[6] Paolo Boldi, Francesco Bonchi, Aris Gionis, and Tamir Tassa. 2012. Injecting uncertainty in graphs for identity obfuscation. *arXiv preprint arXiv:1208.4145* (2012).

[7] Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Antti Ukkonen. 2014. Distance oracles in edge-labeled graphs.. In *EDBT*. 547–558.

[8] Edwin V Bonilla, Kian Chai, and Christopher Williams. 2007. Multi-task Gaussian process prediction. *NeurIPS* 20 (2007).

[9] Hongxu Chen, Hongzhi Yin, Weiqing Wang, Hao Wang, Quoc Viet Hung Nguyen, and Xue Li. 2018. PME: projected metric embedding on heterogeneous networks for link prediction. In *SIGKDD*. 1177–1186.

[10] Yurong Cheng, Ye Yuan, Guoren Wang, Baiyou Qiao, and Zhiqiong Wang. 2014. Efficient sampling methods for shortest path query over uncertain graphs. In *DASFFA*. 124–140.

[11] Yu-Rong Cheng, Ye Yuan, Lei Chen, and Guo-Ren Wang. 2015. Threshold-based shortest path query over large correlated uncertain graphs. *Journal of Computer Science and Technology* 30, 4 (2015), 762–780.

[12] The Koblenz Network Collection. 2017. http://konect.uni-koblenz.de.

[13] Cameron Craddock, Yassine Benhajali, Carlton Chu, Francois Chouinard, Alan Evans, András Jakab, Budhachandra Singh Khundrakpam, John David Lewis, Qingyang Li, Michael Milham, et al. 2013. The neuro bureau preprocessing initiative: open sharing of preprocessed neuroimaging data and derivatives. *Frontiers in Neuroinformatics* 7 (2013), 27.

[14] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative embeddings of latent variable models for structured data. In *ICML*. 2702–2711.

[15] Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. 2007. Transferring naive bayes classifiers for text classification. In *AAAI*, Vol. 7. 540–545.

[16] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. 2008. Self-taught clustering. In *ICML*. 200–207.

[17] Hal Daume III and Daniel Marcu. 2006. Domain adaptation for statistical classifiers. *Journal of artificial Intelligence research* 26 (2006), 101–126.

[18] Jesse Davis and Pedro Domingos. 2009. Deep transfer via second-order markov logic. In *ICML*. 217–224.

[19] Adriana Di Martino, Clare Kelly, Rebecca Grzadzinski, Xi-Nian Zuo, Maarten Mennes, Maria Angeles Mairena, Catherine Lord, F Xavier Castellanos, and Michael P Milham. 2011. Aberrant striatal functional connectivity in children with autism. *Biological psychiatry* 69, 9 (2011), 847–856.

[20] Xingbo Du, Junchi Yan, and Hongyuan Zha. 2019. Joint Link Prediction and Network Alignment via Cross-graph Embedding.. In *IJCAI*. 2251–2257.

[21] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. *NeurIPS* 28 (2015).

[22] David Eppstein. 1998. Finding the k shortest paths. *SIAM Journal on computing* 28, 2 (1998), 652–673.

[23] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective community search over large spatial graphs. *PVLDB* 10, 6 (2017), 709–720.

[24] Kunihiko Fukushima. 1969. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics* 5, 4 (1969), 322–333.

[25] Kunihiko Fukushima and Sei Miyake. 1982. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*. 267–285.

[26] Luis García Domínguez, Jim Stieben, José Luis Pérez Velázquez, and Stuart Shanker. 2013. The imaginary part of coherency in autism: differences in cortical functional connectivity in preschool children. *PLoS One* 8, 10 (2013), e75941.

[27] Joy Ghosh, Hung Q Ngo, Seokhoon Yoon, and Chunming Qiao. 2007. On a routing problem within probabilistic graphs and its application to intermittently connected networks. In *IEEE INFOCOM*. 1721–1729.

[28] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *SIGKDD*. 855–864.

[29] Kai Han, Fei Gui, Xiaokui Xiao, Jing Tang, Yuntian He, Zongmai Cao, and He Huang. 2019. Efficient and effective algorithms for clustering uncertain graphs. *PVLDB* 12, 6 (2019), 667–680.

[30] Jiafeng Hu, Reynold Cheng, Zhipeng Huang, Yixang Fang, and Siqiang Luo. 2017. On embedding uncertain graphs. In *CIKM*. 157–166.
[31] Ming Hua and Jian Pei. 2010. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *EDBT*. 347–358.
[32] Shixun Huang, Zhifeng Bao, J Shane Culpepper, and Bang Zhang. 2019. Finding temporal influential users over evolving social networks. In *ICDE*. 398–409.
[33] Shixun Huang, Zhifeng Bao, Guoliang Li, Yanghao Zhou, and J Shane Culpepper. 2020. Temporal network representation learning via historical neighborhoods aggregation. In *ICDE*. IEEE, 1117–1128.
[34] Shixun Huang, Wenqing Lin, Zhifeng Bao, and Jiachen Sun. 2022. Influence maximization in real-world closed social networks. *PVLDB* 16, 2 (2022), 180–192.
[35] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. 448–456.
[36] Ruoming Jin, Lin Liu, Bolin Ding, and Haixun Wang. 2011. Distance-constraint reachability computation in uncertain graphs. *PVLDB* 4, 9 (2011), 551–562.
[37] Donald B Johnson. 1977. Efficient algorithms for shortest paths in sparse networks. *J. ACM* 24, 1 (1977), 1–13.
[38] Richard M Karp and Michael Luby. 1983. Monte-Carlo algorithms for enumeration and reliability problems. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. 56–64.
[39] Xiangyu Ke, Arijit Khan, Mohammad Al Hasan, and Rojin Rezvansangsari. 2020. Reliability maximization in uncertain graphs. *TKDE* (2020).
[40] Xiangyu Ke, Arijit Khan, and Leroy Lim Hong Quan. 2019. An in-depth comparison of st reliability algorithms over uncertain graphs. *arXiv preprint arXiv:1904.05300* (2019).
[41] Neil D Lawrence and John C Platt. 2004. Learning to learn with the informative vector machine. In *ICML*. 65.
[42] Su-In Lee, Vassil Chatalbashev, David Vickrey, and Daphne Koller. 2007. Learning a meta-level prior for feature relevance from multiple related tasks. In *ICML*. 489–496.
[43] Xiaodong Li, Reynold Cheng, Yixiang Fang, Jiafeng Hu, and Silviu Maniu. 2018. Scalable evaluation of k-NN queries on large uncertain graphs. In *EDBT*.
[44] Chenhao Ma, Reynold Cheng, Laks VS Lakshmanan, Tobias Grubenmann, Yixiang Fang, and Xiaodong Li. 2019. Linc: a motif counting algorithm for uncertain graphs. *PVLDB* 13, 2 (2019), 155–168.
[45] Samuel Madden. 2004. Intel lab data. http://db.csail.mit.edu/labdata/labdata.html.
[46] Lilyana Mihalkova, Tuyen Huynh, and Raymond J Mooney. 2007. Mapping and revising markov logic networks for transfer learning. In *AAAI*, Vol. 7. 608–614.
[47] Lilyana Mihalkova and Raymond J Mooney. 2008. Transfer learning by mapping with minimal target data. In *Proceedings of the AAAI-08 workshop on transfer learning for complex tasks*.
[48] OpenStreetMap contributors. 2017. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org.
[49] Panos Parchas, Francesco Gullo, Dimitris Papadias, and Franceseco Bonchi. 2014. The pursuit of a good possible world: extracting representative instances of uncertain graphs. In *SIGMOD*. 967–978.
[50] Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. 2010. K-nearest neighbors in uncertain graphs. *PVLDB* 3, 1-2 (2010), 997–1008.
[51] Liang Qu, Huaisheng Zhu, Qiqi Duan, and Yuhui Shi. 2020. Continuous-time link prediction via temporal dependent graph neural network. In *Proceedings of The Web Conference*. 3026–3032.
[52] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. 2007. Self-taught learning: transfer learning from unlabeled data. In *ICML*. 759–766.
[53] Arkaprava Saha, Ruben Brokkelkamp, Yllka Velaj, Arijit Khan, and Francesco Bonchi. 2021. Shortest paths and centrality in uncertain networks. *VLDB* 14, 7 (2021), 1188–1201.
[54] Hidetoshi Shimodaira. 2000. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference* 90, 2 (2000), 227–244.
[55] Leslie G Valiant. 1979. The complexity of enumeration and reliability problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.
[56] Zheng Wang, Yangqiu Song, and Changshui Zhang. 2008. Transferred dimensionality reduction. In *Joint European conference on machine learning and knowledge discovery in databases*. 550–565.
[57] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).
[58] Ye Yuan, Lei Chen, and Guoren Wang. 2010. Efficiently answering probability threshold-based shortest path queries over uncertain graphs. In *DASFFA*. 155–170.
[59] Bianca Zadrozny. 2004. Learning and evaluating classifiers under sample selection bias. In *ICML*. 114.
[60] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. 2020. A comprehensive survey on transfer learning. *Proc. IEEE* 109, 1 (2020), 43–76.

[61] Lei Zou, Peng Peng, and Dongyan Zhao. 2011. Top-K possible shortest path query over a large uncertain graph. In *WISE*. 72–86.

Received October 2022; revised January 2023; accepted February 2023