Scalable Probabilistic Routes

Suwei Yang[1,2], Victor C. Liang[2], and Kuldeep S. Meel[1]

[1] National University of Singapore, Singapore
[2] GrabTaxi Holdings, Singapore

Abstract

Inference and prediction of routes have become of interest over the past decade owing to a dramatic increase in package delivery and ride-sharing services. Given the underlying combinatorial structure and the incorporation of probabilities, route prediction involves techniques from both formal methods and machine learning. One promising approach for predicting routes uses decision diagrams that are augmented with probability values. However, the effectiveness of this approach depends on the size of the compiled decision diagrams. The scalability of the approach is limited owing to its empirical runtime and space complexity. In this work, our contributions are two-fold: first, we introduce a relaxed encoding that uses a linear number of variables with respect to the number of vertices in a road network graph to significantly reduce the size of resultant decision diagrams. Secondly, instead of a stepwise sampling procedure, we propose a single pass sampling-based route prediction. In our evaluations arising from a real-world road network, we demonstrate that the resulting system achieves around twice the quality of suggested routes while being an order of magnitude faster compared to state-of-the-art.

1 Introduction

The past decade has witnessed an unprecedented rise of the service economy, best highlighted by the prevalence of delivery and ride-sharing services [9, 10]. For operational and financial efficiency, a fundamental problem for such companies is the inference and prediction of routes taken by the drivers. When a driver receives a job to navigate from location A to B, the ride-sharing app needs to predict the route in order to determine: (1) the trip time, which is an important consideration for the customer, (2) the fare, important consideration for both the driver and the customer, and (3) the trip experience since customers feel safe when the driver takes the route described in their app [2, 35]. However the reality is that drivers and customers have preferences, as such the trips taken are not always the shortest possible by distance or time [22]. To this end, delivery and ride-sharing service companies have a need for techniques that can infer the distribution of routes and efficiently predict the likely route a driver takes for a given start and end location.

Routing, a classic problem in computer science, has traditionally been approached without considering the learning of distributions [1, 30]. However, Choi, Shen, and Darwiche demonstrated through a series of papers that the distribution of routes can be conceptualized as a structured probability distribution (SPD) given the underlying combinatorial structure [6, 31, 32]. Decision diagrams, which are particularly well-suited for representing SPDs, have emerged as the state-of-the-art approach for probability guided routing. The decision diagram based approach allows for learning of SPDs through the use of decision diagrams augmented with probability values, followed by a stepwise process for uncovering the route node by node.

However, scalability remains a challenge when using decision diagrams to reason about route distributions, particularly for large road networks. Existing works address this concern in various ways, such as through the use of hierarchical diagrams [6] and Structured Bayesian Networks [31]. Choi et al. [6] partition the structured space into smaller subspaces, with each

subspace's SPD being represented by a decision diagram. Shen et al. used Structured Bayesian Networks to represent conditional dependencies between sets of random variables, with the distribution within each set of variables represented by a conditional Probabilistic Sentential Decision Diagram (PSDD) [31, 32]. Despite these efforts, the scalability of decision diagrams for routing, in terms of space complexity, remains an open issue [7].

The primary contribution of this work is to tackle the scalability challenges faced by the current state-of-the-art approaches. Our contributions are two-fold: first, we focus on minimizing the size of the compiled diagram by *relaxation and refinement*. In particular, instead of learning distributions over the set of all valid routes, we learn distributions over an over-approximation, perform sampling followed by refinement to output a valid route. Secondly, instead of a stepwise sampling procedure, we perform one-pass sampling by adapting existing sampling algorithm [37] to perform conditional sampling. Our extensive empirical evaluation over benchmarks arising from real-world publicly available road network data demonstrates that our approach, called ProbRoute, is able to handle real-world instances that were clearly beyond the reach of the state-of-the-art. Furthermore, on instances that can be handled by prior state-of-the-art, ProbRoute achieves a median of 10× runtime performance improvements.

2 Background

In the remaining parts of this work we will discuss how to encode simple, more specifically simple trips, in a graph using Boolean formulas. In addition, we will also discuss decision diagrams and probabilistic reasoning with them. In this section, we introduce the preliminaries and background for the rest of the paper. To avoid ambiguity, we use *vertices* to refer to nodes of road network graphs and *nodes* to refer to nodes of decision diagrams.

2.1 Preliminaries

Simple Trip Let G be an arbitrary undirected graph, a path on G is a sequence of connected vertices $v_1, v_2, ..., v_m$ of G where $\forall_{i=1}^{m-1} v_{i+1} \in N(v_i)$, with $N(v_i)$ referring to neighbours of v_i. A path π does not contain loops if $\forall_{v_i,v_j \in \pi} v_i \neq v_j$. π does not contain detour if $\forall_{v_i,v_j,v_k,v_l \in \pi} v_j \notin N(v_i) \vee v_k \notin N(v_i) \vee v_l \notin N(v_i)$. Path π is a simple path if it does not contain loops. A simple path π is a simple trip if it does not contain detours. We denote the set of all simple trips in G as $\mathsf{SimpleTrip}(G)$. In Figure 1, d-e-h is a simple trip whereas d-e-f-c-b-e-h and d-e-f-i-h are not because they contain a loop and a detour respectively. We use $\mathsf{Term}(\pi)$ to refer to the terminal vertices of path π.

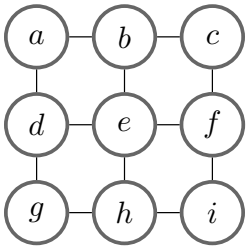

Figure 1: A 3 × 3 grid graph

Probabilistic Routing Problem In this paper, we tackle the probabilistic routing problem which we define as the following. Given a graph G of an underlying road network, training and testing data D_{train}, D_{test}, start and end vertices s, t, sample path π from s to t such that ε-match rate with ground truth path $\pi' \in D_{test}$ is maximized. We define ε-match rate between π and π' as $|U_{close(\pi)}| \div |U|$ where U is the set of vertices of π' and $U_{close(\pi)}$ is the set of vertices of π' that are within ε euclidean distance away from the nearest vertex in π. More details on ε will be discussed in Section 4.

Boolean Formula A Boolean variable can take values *true* or *false*. A literal x is a Boolean variable or its negation. Let F be a Boolean formula. F is in conjunctive normal form (CNF) if F is a conjunction of clauses, where each clause is a disjunction of literals. F is satisfiable if there exists an assignment τ of the set of variables X of F such that F evaluates to *true*. We refer to τ as a satisfying assignment of F and denote the set of all τ as $\mathsf{Sol}(F)$. In the remaining parts of this paper, all formulas and variables are Boolean unless stated otherwise.

Decision Diagrams Decision diagrams are directed acyclic graph (DAG) representations of logical formulas under the knowledge compilation paradigm. Decision diagrams are designed to enable the tractable handling of certain types of queries, that is the queries can be answered in polynomial time with respect to the number of nodes in the diagram [12]. We use diagram size to refer to the number of nodes in a decision diagram. In this work we use the OBDD[$\wedge$] [20] variant of OBDD, more specifically Probabilistic OBDD[$\wedge$] (PROB) [37], for which there are existing efficient sampling algorithm. We will discuss the PROB decision diagram in later sections.

2.2 Related Works

The continuous pursuit of compact representations by the research community has resulted in several decision diagram forms over the years. Some of the decision diagram forms include AOMDD for multi-valued variables, OBDD and SDD for binary variables [5, 11, 24]. Both OBDD and SDD are canonical representations of Boolean formulas given variable ordering for OBDD and *Vtree* for SDD respectively. OBDD [5] is comprised of internal nodes that correspond to variables and leaf nodes that correspond to $\top$ or $\bot$. Each internal node of OBDD have exactly two child and represents the Shannon decomposition [4] on the variable represented by that internal node. SDDs are comprised of elements and nodes [11]. Elements represent conjunction of a *prime* and a *sub*, each of which can either be a terminal SDD or a decomposition SDD. A decomposition SDD is represented by a node with child elements representing the decomposition. A terminal SDD can be a literal, $\top$ or $\bot$. The decompositions of SDDs follow that of the respective *Vtree*, which is a full binary decision tree of Boolean variables in the formula. In this work, we use the OBDD[$\wedge$] [20] variant of OBDD, which is shown to be theoretically incomparable but empirically more succinct than SDDs [20].

A related development formulates probabilistic circuits [8], based on sum-product networks [28] and closely related to decision diagrams, as a special class of neural networks known as Einsum networks [27]. In the Einsum network structure, leaf nodes represent different gaussian distributions. By learning from data, Einsum networks are able to represent SPDs as weighted sums and mixtures of gaussian distributions. Einsum networks address scalability by utilizing tensor operations implemented in existing deep learning frameworks such as PyTorch [26]. Our work differs from the Einsum network structure, we require the determinism property for

decision diagrams whereas the underlying structure for Einsum network lacks this property. We will introduce the determinism property in the following section.

Various Boolean encodings have been proposed for representing paths within a graph, including Absolute, Compact, and Relative encodings [29]. These encodings capture both the vertices comprising the path and the ordering information of said vertices. However, these encodings necessitate the use of polynomial number of Boolean variables, specifically $|V|^2$, $|V|log_2|V|$, and $2|V|^2$ variables for Absolute, Compact, and Relative encoding respectively. While these encodings accurately represent the space of paths within a graph, they are not efficient and lead to high space and time complexity for downstream routing tasks.

Choi, Shen, and Darwiche demonstrated over a series of papers that the distribution of routes can be conceptualized as a structured probability distribution (SPD) given the underlying combinatorial structure [6, 31, 32]. This approach, referred to as the 'CSD' approach in the rest of this paper, builds on top of existing approaches that represents paths using zero-surpressed decision diagrams [19, 17, 18]. The CSD approach utilizes sentential decision diagrams to represent the SPD of paths and employs a stepwise methodology for handling path queries. Specifically, at each step, the next vertex to visit is determined to be the one with the highest probability, given the vertices already visited and the start and destination vertices. While the CSD approach has been influential in its incorporation of probabilistic elements in addressing the routing problem, it is not without limitations. In particular, there are two main limitations (1) there are no guarantees of completion, meaning that even if a path exists between a given start and destination vertex, it may not be returned using the CSD approach [6]. (2) the stepwise routing process necessitates the repeated computation of conditional probabilities, resulting in runtime inefficiencies.

In summary, the limitations of prior works are Boolean encodings that require a high number of variables, lack of routing task completion guarantees, and numerous conditional probability computations.

2.3 PROB: Probabilistic OBDD[∧]

In this subsection, we will introduce the PROB (Probabilistic OBDD[∧]) decision diagram structure and properties. We adopt the same notations as prior work [37] for consistency.

Notations We use *nodes* to refer to nodes in PROB ψ and *vertices* to refer to nodes in graph $G(V, E)$ to avoid ambiguity. $\mathsf{Child}(n)$ refers to the children of node n. $\mathsf{Hi}(n)$ refers to the hi child of decision node n and $\mathsf{Lo}(n)$ refers to the lo child of n. $\theta_{\mathsf{Hi}}(n)$ and $\theta_{\mathsf{Lo}}(n)$ refer to the parameters associated with the edge connecting decision node n with $\mathsf{Hi}(n)$ and $\mathsf{Lo}(n)$ respectively in a PROB. $\mathsf{Var}(n)$ refers to the associated variable of decision node n. $\mathsf{VarSet}(n)$ refers to the set of variables of F represented by a PROB with n as the root node. $\mathsf{Subdiagram}(n)$ refers to the PROB starting at node n. $\mathsf{Parent}(n)$ refers to the immediate parent nodes of n in PROB.

PROB Structure Let ψ be a PROB which represents a Boolean formula F. ψ is a DAG comprising of four types of nodes - conjunction node, decision node, true node, and false node.

A conjunction node (or ∧-node) n_c splits Boolean formula F into different sub-formulas, i.e. $F = F_1 \wedge F_2 \wedge ... \wedge F_j$. Each sub-formula is represented by a PROB rooted at the corresponding child node of n_c, such that the set of variables in each of F_1, F_2, ..., F_j are mutually disjoint.

A decision node n_d corresponds to a Boolean variable x and has exactly two child nodes, $\mathsf{Hi}(n_d)$ and $\mathsf{Lo}(n_d)$. $\mathsf{Hi}(n_d)$ represents the decision to set x to *true* and $\mathsf{Lo}(n_d)$ represents

otherwise. We use $\mathsf{Var}(n_d)$ to refer to the Boolean variable x that decision node n_d is associated with in F. Each branch of n_d has an associated parameter, and the branch parameters of n_d sum up to 1.

The leaf nodes of PROB ψ are true and false nodes. An assignment τ of Boolean formula F is a traversal of the PROB from the root node to the leaf node, we denote such a traversal as $\mathsf{Rep}_\psi(\tau)$. At each decision node n_d, the traversal follows the value of variable $\mathsf{Var}(n_d)$ in τ. At each conjunction node, all child branches are traversed. A satisfying assignment of F will result in a traversal from root to leaf nodes where only the true nodes are visited. If a traversal leads to a false node at any point, then the assignment is not a satisfying assignment.

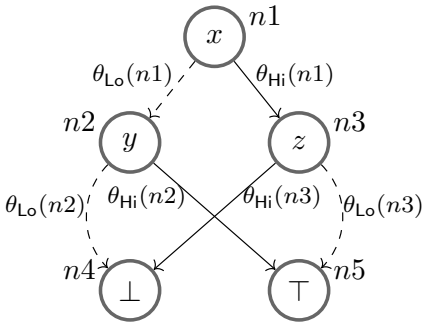

Figure 2: A PROB ψ_1 representing $F = (x \vee y) \wedge (\neg x \vee \neg z)$

An assignment of Boolean formula F is represented by a top-down traversal of a PROB compiled from F. For example, we have a Boolean formula $F = (x \vee y) \wedge (\neg x \vee \neg z)$, represented by the PROB ψ_1 in Figure 2. When x is assigned *true* and z is assigned *false*, F will evaluate to *true*. If we have a partial assignment τ, we can perform inference conditioned on τ if we visit only the branches of decision nodes in ψ that agree with τ. This allows for conditional sampling, which we discuss in Section 3.

PROB inherits important properties of OBDD[$\wedge$] that are useful to our algorithms in later sections. The properties are - *determinism*, *decomposability*, and *smoothness*.

Property 1 (Determinism)**.** *For every decision node n_d, the set of satisfying assignments represented by* $\mathsf{Hi}(n_d)$ *and* $\mathsf{Lo}(n_d)$ *are logically disjoint*

Property 2 (Decomposability)**.** *For every conjunction node n_c,* $\mathsf{VarSet}(c_i) \cap \mathsf{VarSet}(c_j) = \emptyset, \forall c_i, c_j \in \mathsf{Child}(n_c), c_i \neq c_j$

Property 3 (Smoothness)**.** *For every decision node n_d,* $\mathsf{VarSet}(\mathsf{Hi}(n_d)) = \mathsf{VarSet}(\mathsf{Lo}(n_d))$

In the rest of this paper, all mentions of PROB refer to smooth PROB. *Smoothness* can be achieved via a smoothing algorithm introduced in prior work [37]. We defer the smoothing algorithm to the appendix.

3 Approach

In this section, we introduce our approach, ProbRoute, which addresses the aforementioned limitations of existing methods using (1) a novel relaxed encoding that requires a linear number of Boolean variables and (2) a single-pass sampling and refinement approach which provides completeness guarantees. The flow of ProbRoute is shown in Figure 3.

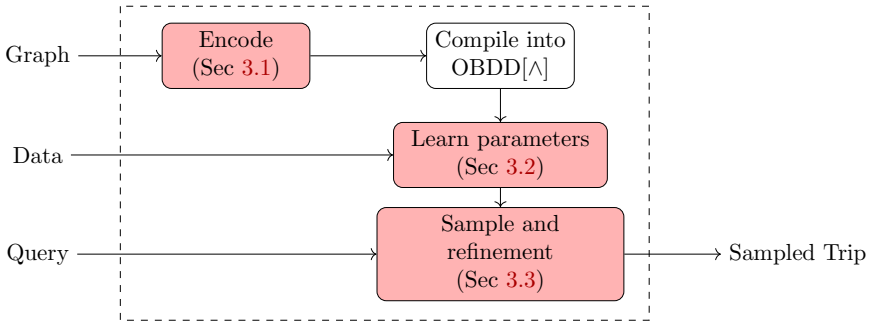

Figure 3: Flow of ProbRoute, with red rectangles indicating this work. For compilation, we use existing off-the-shelf techniques.

In our approach, we first use our relaxed encoding to encode $\mathsf{SimpleTrip}(G)$ of graph G into a Boolean formula. Next, we compile the Boolean formula into OBDD[$\wedge$]. In order to learn from historical trip data, we convert the data into assignments. Subsequently, the OBDD[$\wedge$] is parameterized into PROB ψ and the parameters are learned from data. Finally to sample trips from start vertex v_s to destination vertex v_t, we create a partial assignment τ' with the variables that indicate v_s and v_t are terminal vertices set to *true*. The ProbSample algorithm, algorithm 2, takes τ' as input and samples a set of satisfying assignments. Finally, in the refinement step, a simple trip π is extracted from each satisfying assignment τ using depth-first search to remove disjoint loop components.

3.1 Relaxed Encoding

In this work, we present a novel relaxed encoding that only includes vertex membership and terminal information. Our encoding only requires a linear ($2|V|$) number of Boolean variables, resulting in more succinct decision diagrams and improved runtime performance for downstream tasks. In relation to prior encodings, we observed that the ordering information of vertices can be inferred from the graph given a set of vertices and the terminal vertices, thus enabling us to exclude ordering information in our relaxed encoding. Our relaxed encoding represents an over-approximation of trips in $\mathsf{SimpleTrip}(G)$ for graph $G(V, E)$ using a linear number of Boolean variables with respect to $|V|$. We discuss the over-approximation in later parts of this section.

Our encoding uses two types of Boolean variables, n-type and s-type variables. Each vertex $v \in V$ in graph $G(V, E)$ has a corresponding n-type and s-type variable. The n-type variable indicates if vertex v is part of a trip and s-type variable indicates if v is a terminal vertex of the trip. Our encoding is the conjunction of the five types of clauses over all vertices in graph G as follows.

$$\bigvee_{i \in V} s_i \tag{H1}$$

$$\bigwedge_{i \in V} [n_i \longrightarrow \bigvee_{j \in adj(i)} n_j] \tag{H2}$$

$$\bigwedge_{\substack{i,j,k \in V, \\ i \neq j \neq k}} (\neg s_i \vee \neg s_j \vee \neg s_k) \tag{H3}$$

$$\bigwedge_{i \in V} s_i \longrightarrow n_i \wedge \bigwedge_{j,k \in adj(i), j \neq k} (\neg n_j \vee \neg n_k) \tag{H4}$$

$$\bigwedge_{i,j \in V, j \in adj(i)} [n_i \wedge n_j \longrightarrow s_i \vee [(\bigvee_{\substack{k \in adj(i), \\ k \neq j \neq i}} n_k) \wedge \bigwedge_{\substack{l,m \in adj(i), \\ l,m \neq j}} (\neg n_l \vee \neg n_m)]] \tag{H5}$$

A simple trip π in graph G has at least one terminal vertex and at most two terminal vertices, described by encoding components H1 and H3 respectively. At each terminal vertex v_i of π, there can only be at most 1 adjacent vertex of v_i that is also part of π and this is encoded by H4. For each vertex v_i in π, at least one of their adjacent vertices is in π regardless if v_i is a terminal vertex or otherwise, this is captured by H2. Finally, H5 encodes that if a given vertex v_i and one of its adjacent vertices are part of π, then either another neighbour vertex of v_i is part of π or v_i is a terminal vertex.

Definition 1. *Let* $\mathcal{M} : \mathsf{SimpleTrip}(G) \mapsto \mathsf{Sol}(F)$ *such that for a given trip* $\pi \in \mathsf{SimpleTrip}(G)$, $\tau = \mathcal{M}(\pi)$ *is the assignment whereby the* n*-type variables of all vertices* $v \in \pi$ *and the* s*-type variables of* $v \in \mathsf{Term}(\pi)$ *are set to true. All other variables are set to false in* τ*.*

We refer to our encoding as relaxed encoding because the solution space of constraints over-approximates the space of simple trips in the graph. Notice that while all simple trips correspond to a satisfying assignment of the encoding, they are not the only satisfying assignments. Assignments corresponding to a simple trip π with disjoint loop component β also satisfy the constraints. The intuition is that β introduces no additional terminal vertices, hence H1, H3, and H4 remain satisfied. Since the vertices in β always have n-type variables of exactly two of its neighbours set to *true*, H5 and H2 remain satisfied. Thus, a simple trip with a disjoint loop component also corresponds to a satisfying assignment of our encoding.

3.2 Learning Parameters from Data

We introduce algorithm 1, ProbLearn, for updating branch counters of PROB ψ from assignments. In order to learn branch parameters $\theta_{\mathsf{Hi}}(n)$ and $\theta_{\mathsf{Lo}}(n)$ of decision node n, we require a counter for each of its branches, $\mathsf{Hi}_{\#}(n)$ and $\mathsf{Lo}_{\#}(n)$ respectively. In the learning process, we have a dataset of assignments for Boolean variables in the Boolean formula represented by PROB ψ. For each assignment τ in the dataset, we perform a top-down traversal of ψ following Algorithm 1. In the traversal, we visit all child branches of conjunction nodes (line 4) and the child branch of decision node n corresponding to the assignment of $\mathsf{Var}(n)$ in τ (lines 6 to 12), and increment the corresponding counters in the process. Subsequently, the branch parameters for node n are updated according to the following formulas.

$$\theta_{\mathsf{Hi}}(n) = \frac{\mathsf{Hi}_{\#}(n) + 1}{\mathsf{Hi}_{\#}(n) + \mathsf{Lo}_{\#}(n) + 2} \qquad \theta_{\mathsf{Lo}}(n) = \frac{\mathsf{Lo}_{\#}(n) + 1}{\mathsf{Hi}_{\#}(n) + \mathsf{Lo}_{\#}(n) + 2}$$

Algorithm 1 ProbLearn - updates counters of ψ from data

Input: PROB ψ, τ - complete assignment of data instance

```
1: n ← rootNode(ψ)
2: if n is ∧-node then
3:    for c in Child(n) do
4:       ProbLearn(c, τ)
5: if n is decision node then
6:    l ← getLiteral(τ, Var(n))
7:    if l is positive literal then
8:       Hi#(n) += 1
9:       ProbLearn(Hi(n), τ)
10:   else
11:      Lo#(n) += 1
12:      ProbLearn(Lo(n), τ)
```

While we add 1 to numerator and 2 to denominator as a form of Laplace smoothing [23], other forms of smoothing to account for division by zero is possible. Notice that the learnt branch parameters of node n are in fact approximations of conditional probabilities according to Proposition 1 and Remark 1 as follows.

Proposition 1. *Let* $n1$ *and* $n2$ *be decision nodes where* $n1 = \mathsf{Parent}(n2)$ *and* $\mathsf{Lo}(n1) = n2$, $\theta_{\mathsf{Lo}}(n2) = \frac{\mathsf{Lo}_{\#}(n2)+1}{\mathsf{Lo}_{\#}(n1)+2}$ *and* $\theta_{\mathsf{Hi}}(n2) = \frac{\mathsf{Hi}_{\#}(n2)+1}{\mathsf{Lo}_{\#}(n1)+2}$.

Proof. Recall that the Lo branch parameter of $n2$ is:

$$\theta_{\mathsf{Lo}}(n2) = \frac{\mathsf{Lo}_{\#}(n2)+1}{\mathsf{Hi}_{\#}(n2)+\mathsf{Lo}_{\#}(n2)+2}$$

Notice that $\mathsf{Hi}_{\#}(n2)+\mathsf{Lo}_{\#}(n2) = \mathsf{Lo}_{\#}(n1)$ as all top-down traversals of ψ that pass through $n2$ will have to pass through the Lo branch of $n1$.

$$\theta_{\mathsf{Lo}}(n2) = \frac{\mathsf{Lo}_{\#}(n2)+1}{\mathsf{Lo}_{\#}(n1)+2}$$

A similar argument can be made for $\theta_{\mathsf{Hi}}(n2)$ by symmetry. In the general case if $n2$ has more than one parent, then the term $\mathsf{Hi}_{\#}(n2)+\mathsf{Lo}_{\#}(n2)$ is the sum of counts of branch traversals of all parent nodes of $n2$ that leads to $n2$. Additionally, any conjunction node c between $n1$ and $n2$ will not affect the proof because all children of c will be traversed. For understanding, one can refer to the example in Figure 2 where $n1$ corresponds to the root node.

□

Remark 1. *Recall that* $\mathsf{Var}(n1) = x$ *and* $\mathsf{Var}(n2) = y$ *in* PROB ψ_1 *in Figure 2. Observe that* $\frac{\mathsf{Lo}_{\#}(n2)}{\mathsf{Lo}_{\#}(n1)}$ *for* PROB ψ_1 *in Figure 2 is the conditional probability* $Pr(\neg y|\neg x)$ *as it represents the count of traversals that passed through Lo branch of* $n2$ *out of total count of traversals that passed through Lo branch of* $n1$. *A similar observation can be made for* $\mathsf{Hi}(n2)$.

Notice that as the $\mathsf{Lo}_{\#}(n2)$ *and* $\mathsf{Lo}_{\#}(n1)$ *becomes significantly large, that is* $\mathsf{Lo}_{\#}(n2) >> 1$ *and* $\mathsf{Lo}_{\#}(n1) >> 2$*:*

$$\theta_{\mathsf{Lo}}(n2) = \frac{\mathsf{Lo}_{\#}(n2)+1}{\mathsf{Lo}_{\#}(n1)+2} \approx \frac{\mathsf{Lo}_{\#}(n2)}{\mathsf{Lo}_{\#}(n1)} = Pr(\neg y|\neg x)$$

As such, the learnt branch parameters are approximate conditional probabilities.

3.3 Sampling Trip Query Answers

Algorithm 2 ProbSample - returns sampled assignment

Input: PROB ψ, partial assignment τ'
Output: complete assignment that agrees with τ'

caches $\omega, \gamma \longleftarrow$ initCache()
for node n in bottom-up ordering of ψ **do**
  if n is $\top$ node **then**
    $\omega[n] \longleftarrow \emptyset$, $\gamma[n] \longleftarrow 1$
  else if n is $\bot$ node **then**
    $\omega[n] \longleftarrow$ Invalid, $\gamma[n] \longleftarrow 0$
  else if n is $\wedge$ node **then**
    $\omega[n] \longleftarrow \mathsf{unionChild}(\mathsf{Child}(n), \omega)$
    $\gamma[n] \longleftarrow \prod_{c \in \mathsf{Child}(n)} \gamma[c]$
  else
    if $\mathsf{Var}(n)$ in τ' **then**
      if $\omega[\tau'[\mathsf{Var}(n)]]$ is Invalid **then**
        $\omega[n] \longleftarrow$ Invalid, $\gamma[n] \longleftarrow 0$
      else
        $\omega[n] \longleftarrow \mathsf{followAssign}(\tau)$
        if $\tau'[\mathsf{Var}(n)]$ is $\neg\mathsf{Var}(n)$ **then**
          $\gamma[n] \longleftarrow \theta_{\mathsf{Lo}}(n) \times \gamma[\mathsf{Lo}(n)]$
        else
          $\gamma[n] \longleftarrow \theta_{\mathsf{Hi}}(n) \times \gamma[\mathsf{Hi}(n)]$
    else
      $l \longleftarrow \theta_{\mathsf{Lo}}(n) \times \gamma[\mathsf{Lo}(n)]$
      $h \longleftarrow \theta_{\mathsf{Hi}}(n) \times \gamma[\mathsf{Hi}(n)]$
      $\gamma[n] \longleftarrow l + h$
      $\alpha \longleftarrow \mathsf{Binomial}(\frac{h}{l+h})$
      if α is 1 **then**
        $\omega[n] \longleftarrow \omega[\mathsf{Hi}(n)] \cup \mathsf{Var}(n)$
      else
        $\omega[n] \longleftarrow \omega[\mathsf{Lo}(n)] \cup \neg\mathsf{Var}(n)$
return ω[rootnode(ψ)]

The ability to conditionally sample trips is critical to handling trip queries for arbitrary start-end vertices, for which a trip is to be sampled conditioned on the given start and end vertices. In this work, we adapted the weighted sampling algorithm using PROB, which was introduced by prior work [37], to support conditional sampling and denote it as ProbSample.

Algorithm 2, ProbSample, performs sampling of satisfying assignments from a PROB ψ in a bottom-up manner. ProbSample takes an input PROB ψ and partial assignment τ' and returns a sampled complete assignment that agrees with τ'. The input τ' specifies the terminal vertices for a given trip query by assigning the s-type variables. ProbSample employs two caches ω and γ, for partially sampled assignment at each node and joint probabilities during the sampling process. In the process, ProbSample performs calculations of joint probabilities at each node. In addition, ProbSample stores the partial samples at each node in ω. The partial sample for a false node would be Invalid as it means that an assignment is unsatisfiable. On the other hand, the partial sample for a true node is $\emptyset$ which will be incremented with variable assignments during the processing of internal nodes of ψ. The partially sampled assignment at every $\wedge$-node c is the union of the samples of all its child nodes, as the child nodes have mutually disjoint variable sets due to *decomposability* property. For a decision node d, if $\mathsf{Var}(d)$ is in τ', the partial sample at d will be the union of the literal in τ' and the partial sample at the corresponding child node (lines 11 to 19) to condition on τ'. Otherwise, the partial assignment at d is sampled according to the weighted joint probabilities l and h (lines 21 to 28). Finally, the output of ProbSample would be the sampled assignment at the root node of ψ. To extend ProbSample to sample k complete assignments, one has to keep k partial assignments in ω at each node during the sampling process and sample k independent partial assignments at each decision node.

Proposition 2. *Let* PROB ψ *represent Boolean formula* F, ProbSample *samples* $\tau \in \mathsf{Sol}(F)$ *according to the joint branch parameters, that is* $\prod_{n\in\mathsf{Rep}_\psi(\tau)}[(1-I_n)\theta_{\mathsf{Lo}}(n)+I_n\theta_{\mathsf{Hi}}(n)]$ *where* I_n *is 1 if* $\mathsf{Hi}(n) \in \mathsf{Rep}_\psi(\tau)$ *and 0 otherwise.*

Proof. Let ψ be a PROB that only consists of decision nodes as internal nodes. At each decision node d in the bottom-up sampling pass, assignment of $\mathsf{Var}(d)$ is sampled proportional to $\theta_{\mathsf{Lo}}(d) \times \gamma[\mathsf{Lo}(d)]$ and $\theta_{\mathsf{Hi}}(d) \times \gamma[\mathsf{Hi}(d)]$ to be *false* and *true* respectively. Without loss of generality, we focus on the term $\theta_{\mathsf{Lo}}(d) \times \gamma[\mathsf{Lo}(d)]$, a similar argument would follow for the other branch by symmetry.

Let $d2$ denote $\mathsf{Lo}(d)$. Notice that $\gamma[d2]$ is $\theta_{\mathsf{Lo}}(d2)\times\gamma[\mathsf{Lo}(d2)]+\theta_{\mathsf{Hi}}(d2)\times\gamma[\mathsf{Hi}(d2)]$. Expanding the equation, the probability of sampling $\neg\mathsf{Var}(d)$ is $\theta_{\mathsf{Lo}}(d) \times \theta_{\mathsf{Lo}}(d2) \times \gamma[\mathsf{Lo}(d2)] + \theta_{\mathsf{Lo}}(d) \times \theta_{\mathsf{Hi}}(d2) \times \gamma[\mathsf{Hi}(d2)]$. If we expand $\gamma[\mathsf{Lo}(d)]$ recursively, we are considering all possible satisfying assignments of $\mathsf{VarSet}(\mathsf{Lo}(d))$, more specifically we would be taking the sum of the product of corresponding branch parameters of each possible satisfying assignment of $\mathsf{VarSet}(\mathsf{Lo}(d))$.

Observe that $\mathsf{Var}(d)$ is sampled to be assigned *false* with probability $\theta_{\mathsf{Lo}}(d) \times \theta_{\mathsf{Lo}}(d2) \times \gamma[\mathsf{Lo}(d2)] + \theta_{\mathsf{Lo}}(d) \times \theta_{\mathsf{Hi}}(d2) \times \gamma[\mathsf{Hi}(d2)]$. Similarly, $\mathsf{Var}(d2)$ is sampled to be assigned *false* with probability $\theta_{\mathsf{Lo}}(d2) \times \gamma[\mathsf{Lo}(d2)]$. Notice that if we view the bottom-up process in reverse, the probability of sampling $\neg\mathsf{Var}(d)$ and $\neg\mathsf{Var}(d2)$ is $\theta_{\mathsf{Lo}}(d)\times\theta_{\mathsf{Lo}}(d2)\times\gamma[\mathsf{Lo}(d2)]$. In the general case, it then follows that a satisfying assignment would reach the *true* node which has γ value set to 1. It then follows that for each $\tau \in \mathsf{Sol}(F)$, τ is sampled with probability $P = \prod_{n\in\mathsf{Rep}_\psi(\tau)}[(1 - I_n)\theta_{\mathsf{Lo}}(n) + I_n\theta_{\mathsf{Hi}}(n)]$. Notice that $\wedge$-nodes have no impact on the sampling probability as no additional terms are introduced in the product of branch parameters. □

Remark 2. *Recall in Remark 1 that* $\theta_{\mathsf{Hi}}(n)$ *and* $\theta_{\mathsf{Lo}}(n)$ *are approximately conditional probabilities. By Proposition 2, assignment* $\tau \in \mathsf{Sol}(F)$ *is sampled with probability proportional to* $\prod_{n\in\mathsf{Rep}_\psi(\tau)}[(1-I_n)\theta_{\mathsf{Lo}}(n)+I_n\theta_{\mathsf{Hi}}(n)]$. *Notice that if we rewrite the product of branch parameters as the product of approximate conditional probability, it is approximately the joint probability of sampling* τ.

Refinement In the refinement step, we extract a trip from sampled assignment τ by removing spurious disjoint loop components using depth-first search.

Definition 2. *Let $\mathcal{M}' : \mathsf{Sol}(F) \mapsto \mathsf{SimpleTrip}(G)$ be the mapping function of the refinement process, for a given graph G and its relaxed encoding F. For an assignment $\tau \in \mathsf{Sol}(F)$, let V_τ be the set of vertices in G that have their n-type variables set to true in τ. A depth-first search is performed from the starting vertex on V_τ, removing disjoint components. The resultant simple path is $\pi = \mathcal{M}'(G)$.*

Although $\mathcal{M}'(\cdot)$ is a many-to-one (i.e. surjective) mapping function, it is not a concern in practice as trips with disjoint loop components are unlikely to occur in real-world or synthetic trip data from which probabilities can be learned.

Theorem 1. *Given $v_s, v_t \in G$, let $\pi_{s,t} \in \mathsf{SimpleTrip}(G)$ be a trip between v_s and v_t. Let $R_{\pi_{s,t}} = \{\tau \mid (\tau \in \mathsf{Sol}(F)) \wedge (\mathcal{M}'(\tau) = \pi_{s,t})\}$. Then,*

$$\Pr[\pi_{s,t} \text{ is sampled}] \propto \sum_{\tau \in R_{\pi_{s,t}}} \prod_{n \in \mathsf{Rep}_\psi(\tau)} [(1 - I_n)\theta_{\mathsf{Lo}}(n) + I_n \theta_{\mathsf{Hi}}(n)]$$

Proof. From Definition 1 and 2, one can say that given a graph G and its relaxed encoding F, $\forall \pi \in \mathsf{SimpleTrip}(G), \exists \tau \in \mathsf{Sol}(F)$ such that $\mathcal{M}'(\tau) = \pi$.

Notice that sampling $\pi_{s,t}$ amounts to sampling $\tau \in R_{\pi_{s,t}}$. As such, the probability of sampling $\pi_{s,t}$ would be the sum over probability of sampling each member of $R_{\pi_{s,t}}$. Recall that the probability of sampling a single assignment τ is proportional to $\prod_{n \in \mathsf{Rep}_\psi(\tau)}[(1 - I_n)\theta_{\mathsf{Lo}}(n) + I_n \theta_{\mathsf{Hi}}(n)]$ by Proposition 2. As such the probability $Pr[\pi_{s,t}$ is sampled$]$ is proportional to $\sum_{\tau \in R_{\pi_{s,t}}} \prod_{n \in \mathsf{Rep}_\psi(\tau)}[(1 - I_n)\theta_{\mathsf{Lo}}(n) + I_n \theta_{\mathsf{Hi}}(n)]$. □

Remark 3. *It is worth noting that $Pr[\pi_{s,t}$ is sampled$] > 0$, as all branch parameters are greater than 0 by definition. Recall that branch parameters are computed with a '+1' in numerator and '+2' in denominator, and given that branch counters are 0 or larger, branch parameters are strictly positive.*

4 Experiments

In order to evaluate the efficacy of ProbRoute, we built a prototype in Python 3.8 with NumPy [16], toposort [33], OSMnx[3], and NetworkX [15] packages. We employ KCBox tool[1] for OBDD[$\wedge$] compilation [20]. The experiments were conducted on a cluster of machines with Intel Xeon Platinum 8272CL processors and 64GB of memory. In the experiments, we evaluated ProbRoute against an adaptation of the state-of-the-art probabilistic routing approach [6] and an off-the-shelf non-probabilistic routing library, Pyroutelib3 [36], in terms of quality of trip suggestions and runtime performance. In particular, we adapted the state-of-the-art approach by Choi et al [6] to sample for trips instead of computing the most probable trip and refer to the adapted approach as 'CSD' in the rest of the section. In addition, we compared our relaxed encoding to existing path encodings across various graphs, specifically to absolute encoding and compact encoding [29].

Through the experiments, we investigate the following:

R1 Can ProbRoute effectively learn from data and sample quality trips?

[1] https://github.com/meelgroup/KCBox

R2 How efficient is our relaxed encoding technique?

R3 What is the runtime performance of the ProbRoute?

Data Generation In this study, we use the real-world road network of Singapore obtained from OpenStreetMap [25] using OSMnx. The road network graph G_r consisted of 23522 vertices and 45146 edges along with their lengths. In addition, we also use an abstracted graph[2] of G_r which we denote as G_a for the remaining of this section. A vertex in G_a corresponds to a unique subgraph of G_r.

Synthetic trips were generated by deviating from shortest path given start and end vertices. A random pair of start and end vertices were selected in G_r and the shortest path π was found. Next, the corresponding intermediate regions of π in G_a are blocked in G_r, and a new shortest path π' was found and deemed to be the synthetic trip generated. We generated 11000 synthetic trips, 10000 for training and 1000 for evaluation. While we used G_a to keep the trip sampling time reasonable, it is possible to use more fine-grained regions for offline applications.

4.1 R1: ProbRoute's Ability to Learn Probabilities

To understand ProbRoute's ability to learn probabilities from data, we evaluate its ability to produce trips that closely resembles the ground truth. Both ProbRoute and CSD, which are sampling-based approaches, were evaluated by sampling 20 trips and taking the median match rate for each instance. Recall that the ε-match rate is defined as the proportion of vertices in the ground truth trip that were within ε meters of euclidean distance from the closest vertex in the proposed trip. In the evaluation, we set the ε tolerance to be the median edge length of G_r, which is 64.359 meters, to account for parallel trips. To further emphasize the advantages of probabilistic approaches, we included an off-the-shelf routing library, Pyroutelib3 [36], in the comparison.

In order to conduct a fair comparison, we have adapted the CSD approach to utilize PROB derived from our relaxed encoding. Our evaluation utilizes this adapted approach to sample a trip on G_a in a stepwise manner, where the probability of the next step is conditioned on the previous step and destination. The conditional probabilities are computed in a similar manner to the computations of joint probabilities, which are the γ cache values, in the ProbSample. The predicted trip on the road network G_r is determined by the shortest trip on the subgraph formed by the sequence of sampled regions. In contrast, ProbRoute samples a trip on G_a in a single pass, and subsequently retrieves the shortest trip on the subgraph of the sampled regions as the predicted trip on G_r. It is worth noting that for sampling-based approaches, there may be instances where a trip cannot be found on G_r due to factors such as a region in G_a containing disconnected components. We incorporated a rejection sampling process with a maximum of 400 attempts and 5 minutes to account for such scenarios.

Table 1 shows the match rate statistics of the respective methods. Under ε-Match setting, where ε is set as the median edge length of G_r to account for parallel trips, ProbRoute has the highest match rate among the three approaches. In addition, ProbRoute produced perfect matches for more than 25% of instances. ProbRoute has 0.316 ε-match rate on median, significantly more than 0.172 for CSD and 0.107 for Pyroutelib. The trend is similar for exact matches, where ε is set to 0 as shown under the 'Exact Match' columns in Table 1. In the exact match setting, ProbRoute achieved a median of 0.310 match rate, almost double that

[2] We use the geohash system (geohash level 5) of partitioning the road network graph. For more information on the format http://geohash.org/site/tips.html#format

Stats	Exact Match			ε-Match		
	Pyroutelib	CSD	ProbRoute	Pyroutelib	CSD	ProbRoute
25%	0.045	0.049	**0.082**	0.061	0.066	**0.102**
50%	0.088	0.160	**0.310**	0.107	0.172	**0.316**
75%	0.185	0.660	**1.000**	0.208	0.663	**1.000**
Mean	0.151	0.359	**0.445**	0.171	0.372	**0.456**

Table 1: Match rate statistics for completed benchmark instances by respective methods. The percentages under 'Stats' column refer to the corresponding percentiles. 'Exact Match' refers to match rate when $\varepsilon = 0$, and 'ε-Match' refers to match rate when ε is set to median edge length of G_r.

of CSD's 0.160 median match rate. The evaluation results also demonstrate the usefulness of probabilistic approaches such as ProbRoute, especially in scenarios where experienced drivers navigate according to their own heuristics which may be independent of the shortest trip. In particular, ProbRoute would be able to learn and suggest trips that align with the unknown heuristics of driver preferences given start and destination locations. Thus, the results provide an affirmative answer to **R1**.

4.2 R2: Efficiency of Relaxed Encoding

Encoding	Grid				SGP
	2	3	4	5	
Absolute	99	1500	31768	1824769	TO
Compact	771	TO	TO	TO	TO
Relaxed(Ours)	**31**	**146**	**2368**	**20030**	**38318**

Table 2: Comparison of OBDD[$\wedge$] size for different graphs, with 3600s timeout. Grid 2 refers to a 2x2 grid graph. SGP refers to abstract graph (G_a) of Singapore road network.

We compared our relaxed encoding to existing path encodings across various graphs, specifically to absolute encoding and compact encoding [29]. In the experiment, we had to adapt compact encoding to CNF form with Tseitin transformation [34], as CNF is the standard input for compilation tools. We compiled the CNFs of the encodings into OBDD[$\wedge$] form with 3600s compilation timeout and compared the size of resultant diagrams. The results are shown in Table 2, with rows corresponding to the different encodings used and columns corresponding to different graphs being encoded. Entries with *TO* indicate that the compilation has timed out. Table 2 shows that our relaxed encoding consistently results in smaller decision diagrams, up to 91$\times$ smaller. It is also worth noting that relaxed encoding is the only encoding that leads to compilation times under 3600s for the abstracted *Singapore* graph. The results strongly support our claims about the significant improvements that our relaxed encoding brings.

4.3 R3: ProbRoute's Runtime Performance

For wide adoption of new routing approaches, it is crucial to be able to handle the runtime demands of existing applications. As such, we measured the relative runtimes of probabilistic approaches, that is ProbRoute and CSD, with respect to existing routing system Pyroutelib

Stats	$\frac{\text{CSD}}{\text{Pyroutelib}} \times 10^3$	$\frac{\text{ProbRoute}}{\text{Pyroutelib}} \times 10^3$
25%	6.33	**1.40**
50%	21.64	**2.00**
75%	47.90	**3.03**
Mean	36.16	**2.62**

Table 3: Relative runtime statistics (lower is better) for completed instances by CSD and ProbRoute. Under column '$\frac{\text{CSD}}{\text{Pyroutelib}}$' and row '50%', CSD approach takes a median of 21.64×10^3 times the runtime of Pyroutelib.

and show the relative runtimes in Table 3. From the result, ProbRoute is more than one order of magnitude faster on median than the existing probabilistic approach CSD. The result also shows that ProbRoute is also on median more than a magnitude closer to Pyroutelib's runtime using the same PROB as compared to CSD approach. In addition, CSD approach timed out on 650 of the 1000 test instances, while ProbRoute did not time out. Additionally, as mentioned in [6], CSD does not guarantee being able to produce a complete trip from start to destination. The results in Table 3 highlight the progress made by ProbRoute in closing the gap between probabilistic routing approaches and existing routing systems.

5 Conclusion

Whilst we have demonstrated the efficiency of our approach, there are possible extensions to make our approach more appealing for wide adoption. In terms of runtime performance, our approach is three orders of magnitude slower than existing probability agnostic routing systems. As such, there is still room for runtime improvements for our approach to be functional replacements of existing routing systems. Additionally, our relaxed encoding only handles undirected graphs at the moment and it would be of practical interest to extend the encoding to directed graphs to handle one-way streets. Furthermore, it would also be of interest to incorporate ideas to improve runtime performance from existing hierarchical path finding algorithms such as contractual hierarchies, multi-level dijkstra and other related works [13, 14, 21].

In summary, we focused on addressing the scalability barrier for reasoning over route distributions. To this end, we contribute two techniques: a relaxation and refinement approach that allows us to efficiently and compactly compile routes corresponding to real-world road networks, and a one-pass route sampling technique. We demonstrated the effectiveness of our approach on a real-world road network and observed around 91× smaller PROB, 10× faster trip sampling runtime and almost 2× the match rate of state-of-the-art probabilistic approach, bringing probabilistic approaches closer to achieving comparable runtime to traditional routing tools.

Acknowledgments

We sincerely thank Yong Lai for the insightful discussions. We sincerely thank reviewers for the constructive feedback. Suwei Yang is supported by the Grab-NUS AI Lab, a joint collaboration between GrabTaxi Holdings Pte. Ltd., National University of Singapore, and the Industrial Postgraduate Program (Grant: S18-1198-IPP-II) funded by the Economic Development Board of Singapore. Kuldeep S. Meel is supported in part by National Research Foundation Singa-

pore under its NRF Fellowship Programme(NRF-NRFFAI1-2019-0004), Ministry of Education Singapore Tier 2 grant (MOE-T2EP20121-0011), and Ministry of Education Singapore Tier 1 Grant (R-252-000-B59-114).

References

[1] Ravindra K Ahuja, Kurt Mehlhorn, James Orlin, and Robert E Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM (JACM)*, 37(2):213–223, 1990.

[2] Siddhartha Banerjee, Carlos Riquelme, and R. Johari. Pricing in ride-share platforms: A queueing-theoretic approach. *Econometrics: Econometric & Statistical Methods - Special Topics eJournal*, 2015.

[3] Geoff Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Econometrics: Computer Programs & Software eJournal*, 2017.

[4] George Boole. An investigation of the laws of thought: On which are founded the mathematical theories of logic and probabilities. 1854.

[5] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, 1986.

[6] Arthur Choi, Yujia Shen, and Adnan Darwiche. Tractability in structured probability spaces. In *NeurIPS*, volume 30, pages 3477–3485, 2017.

[7] Arthur Choi, Guy Van den Broeck, and Adnan Darwiche. Probability distributions over structured spaces. In *AAAI*, 2015.

[8] YooJung Choi, A. Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. 2020.

[9] Regina R. Clewlow and G. Mishra. Disruptive transportation: The adoption, utilization, and impacts of ride-hailing in the united states. 2017.

[10] Jack Collison. The impact of online food delivery services on restaurant sales. 2020.

[11] Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *IJCAI*, 2011.

[12] Adnan Darwiche and P. Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002.

[13] Daniel Delling, Andrew Goldberg, Thomas Pajor, and Renato Werneck. Customizable route planning. In *Proceedings of the 10th International Symposium on Experimental Algorithms*, 2011.

[14] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transp. Sci.*, 2012.

[15] Aric A. Hagberg, Daniel A. Schult, and Pieter Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, 2008.

[16] C. Harris, K. J. Millman, S. Walt, Ralf Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. Smith, R. Kern, Matti Picus, S. Hoyer, M. Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'io, Mark Wiebe, P. Peterson, Pierre G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, Christoph Gohlke, and T. E. Oliphant. Array programming with numpy. *Nature*, 585:357 – 362, 2020.

[17] Takeru Inoue, Hiroaki Iwashita, Jun Kawahara, and Shin ichi Minato. Graphillion: software library for very large sets of labeled graphs. *International Journal on Software Tools for Technology Transfer*, 2016.

[18] Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin ichi Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 2017.

[19] Donald Ervin Knuth. The art of computer programming, volume 4, fascicle 2: Generating all tuples and permutations. 2005.

[20] Yong Lai, Dayou Liu, and Minghao Yin. New canonical representations by augmenting obdds with conjunctive decomposition. *Journal of Artificial Intelligence Research*, 58:453–521, 2017.

[21] Ken C. K. Lee, Wang-Chien Lee, Baihua Zheng, and Yuan Tian. Road: A new spatial object search framework for road networks. *IEEE Transactions on Knowledge and Data Engineering*, 2012.

[22] J. Letchner, John Krumm, and E. Horvitz. Trip router with individualized preferences (trip): Incorporating personalization into route planning. In *AAAI*, 2006.

[23] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. 2008.

[24] R. Mateescu, R. Dechter, and Radu Marinescu. And/or multi-valued decision diagrams (aomdds) for graphical models. *J. Artif. Intell. Res.*, 2008.

[25] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org, 2017.

[26] Adam Paszke, S. Gross, Francisco Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, Alban Desmaison, Andreas Köpf, E. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, B. Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[27] Robert Peharz, Steven Lang, A. Vergari, Karl Stelzner, Alejandro Molina, M. Trapp, Guy Van den Broeck, K. Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *ICML*, 2020.

[28] Hoifung Poon and Pedro M. Domingos. Sum-product networks: A new deep architecture. *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690, 2011.

[29] S. Prestwich. Sat problems with chains of dependent variables. *Discret. Appl. Math.*, 130:329–350, 2003.

[30] Daniel J Rosenkrantz, Richard Edwin Stearns, and Philip M Lewis. Approximate algorithms for the traveling salesperson problem. In *15th Annual Symposium on Switching and Automata Theory (swat 1974)*, pages 33–42. IEEE, 1974.

[31] Yujia Shen, Arthur Choi, and Adnan Darwiche. Conditional psdds: Modeling and learning with modular knowledge. In *AAAI*, 2018.

[32] Yujia Shen, Anchal Goyanka, Adnan Darwiche, and Arthur Choi. Structured bayesian networks: From inference to learning with routes. In *AAAI*, 2019.

[33] Eric V. Smith. toposort, 2022.

[34] G. S. Tseitin. On the complexity of derivation in propositional calculus. 1983.

[35] Zheng Wang, Kun Fu, and Jieping Ye. Learning to estimate the travel time. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[36] Oliver White and Mikolaj Kuranowski. Pyroutelib3, 2017.

[37] Suwei Yang, Victor Liang, and Kuldeep S. Meel. Inc: A scalable incremental weighted sampler. In *FMCAD 2022*, page 205, 2022.

Appendix

Smoothing

Algorithm 3 Smooth - performs smoothing on a PROB

Input: PROB ψ

1: **for** node n in bottom-up pass of ψ **do**
2: **if** n is $\top$ or $\bot$ node **then**
3: $\mathsf{VarSet}(n) \longleftarrow \emptyset$
4: **else if** n is $\wedge$ node **then**
5: $\mathsf{VarSet}(n) \longleftarrow \bigcup_{n_c \in \mathsf{Child}(n)} \mathsf{VarSet}(n_c)$
6: **else**
7: $\delta \longleftarrow \mathsf{createDecisionNodes}(\mathsf{VarSet}(\mathsf{Lo}(n)) \setminus \mathsf{VarSet}(\mathsf{Hi}(n)))$
8: $c \longleftarrow \mathsf{conjunctionNode}()$
9: $\mathsf{Child}(c) \longleftarrow \{\mathsf{Hi}(n) \cup \delta\}$
10: $\mathsf{Hi}(n) \longleftarrow c$
11: $\delta \longleftarrow \mathsf{createDecisionNodes}(\mathsf{VarSet}(\mathsf{Hi}(n)) \setminus \mathsf{VarSet}(\mathsf{Lo}(n)))$
12: $c \longleftarrow \mathsf{conjunctionNode}()$
13: $\mathsf{Child}(c) \longleftarrow \{\mathsf{Lo}(n) \cup \delta\}$
14: $\mathsf{Lo}(n) \longleftarrow c$
15: $\mathsf{VarSet}(n) \longleftarrow \mathsf{VarSet}(\mathsf{Lo}(()n)) \cup \mathsf{VarSet}(\mathsf{Hi}(n))$

An important property to enable a PROB to learn the correct distribution is smoothness. A non-smooth PROB could be missing certain parameters. An example would be if we have an assignment $\tau_1 = [\neg x, y, -z]$, ψ_1 in Figure 2 (main paper) will not have a counter for $\neg z$ as the traversal ends after reaching the true node from the decision node representing variable y. Observe that the above-mentioned issue would not occur in a smooth PROB ψ_2 in Figure 4. If a PROB is not smooth, we can perform smoothing. For a given decision node n (for consistency with algorithm 3), if $\mathsf{VarSet}(\mathsf{Lo}(n)) \setminus \mathsf{VarSet}(\mathsf{Hi}(n)) \neq \emptyset$ we can augment the hi branch of n with the missing variables as shown in lines 7 - 10 of algorithm 3. We create a new conjunction node c with decision nodes representing each missing variable in $\mathsf{VarSet}(\mathsf{Lo}(n)) \setminus \mathsf{VarSet}(\mathsf{Hi}(n))$ and $\mathsf{Hi}(n)$ as children. For each additional decision node created for $\mathsf{VarSet}(\mathsf{Lo}(n)) \setminus \mathsf{VarSet}(\mathsf{Hi}(n))$, both branches lead to the true node. We reassign $\mathsf{Hi}(n)$ to be c. Similarly, we repeat the operation for $\mathsf{Lo}(n)$. Once the smoothing operation is performed on every decision node in ψ, ψ will have the smoothness property.

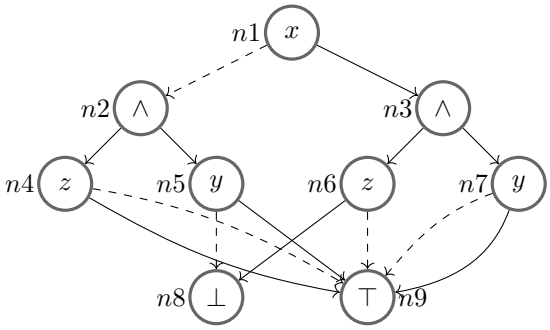

Figure 4: A smooth PROB ψ_2 with 9 nodes, $n1, ..., n9$, representing $F = (x \vee y) \wedge (\neg x \vee \neg z)$. Branch parameters are omitted

Probability Computation

Algorithm 4 ComputeProb - returns probability of τ

Input: PROB ψ, Assignment τ
Output: probability of τ

```
1: cache γ ⟵ initCache()
2: for node n in bottom-up pass of ψ do
3:   if n is ⊤ node then
4:     γ[n] ⟵ 1
5:   else if n is ⊥ node then
6:     γ[n] ⟵ 0
7:   else if n is ∧ node then
8:     γ[n] ⟵ ∏_{c∈Child(n)} γ[c]
9:   else
10:    if Var(n) in τ then
11:      γ[n] ⟵ assignProb(θ_Hi(n), θ_Lo(n), γ)
12:    else
13:      γ[n] ⟵ θ_Lo(n) × γ[Lo(n)] + θ_Hi(n) × γ[Hi(n)]
14: return γ[rootnode(ψ)]
```

Probability computations are typically performed on decision diagrams in a bottom-up manner, processing child nodes before parent nodes. In this work, we implement a probability computation algorithm (ComputeProb), which is the joint probability computation component of ProbSample in the main paper. In line 11, the γ cache value is the product of branch parameter and child γ cache value of the corresponding assignment of $\mathsf{Var}(n)$ instead of both possible possible assignments in line 13.

Additional results

We show in Table 4 additional ε-match rate statistics on how well ProbRoute performs when provided different amount of data to learn probabilities. As we increase the amount of data provided for learning, in increments of 2000 instances (20% of 10000 total), there is a general improvement in the match rate of the trips produced by ProbRoute. A similar trend is observed when $\varepsilon = 0$, with corresponding stats shown in Table 5.

Stats	ProbRoute					
	0%	20%	40%	60%	80%	100%
Mean	0.210	0.416	0.434	0.451	0.466	0.456
Std	0.192	0.360	0.373	0.376	0.383	0.386
25%	0.081	0.102	0.095	0.098	0.098	0.102
50%	0.149	0.286	0.297	0.318	0.349	0.316
75%	0.257	0.715	0.854	0.964	1.000	1.000

Table 4: ε-match rate statistics for ProbRoute where ε is set as median edge length of road network graph G_r. The percentages under 'Stats' column refer to the percentiles, for example '25%' row refer to the 25th percentile match rate for various methods. The percentages under ProbRoute header indicates the percentage of data that ProbRoute has learned from, out of the 10000 learning instances in total.

Stats	ProbRoute					
	0%	20%	40%	60%	80%	100%
Mean	0.192	0.404	0.422	0.440	0.455	0.445
Std	0.192	0.365	0.379	0.382	0.389	0.391
25%	0.063	0.080	0.075	0.078	0.077	0.082
50%	0.132	0.275	0.282	0.308	0.334	0.305
75%	0.243	0.702	0.848	0.963	1.000	1.000

Table 5: Exact match rate statistics for ProbRoute where $\varepsilon = 0$. The percentages under 'Stats' column refer to the percentiles, for example '25%' row refer to the 25th percentile match rate for various methods. The percentages under ProbRoute header indicates the percentage of data that ProbRoute has learned from, out of the 10000 learning instances in total.