Optimal route planning for stochastic time-dependent hitchhiker's problem

Article *in* Journal of Location Based Services · November 2019

DOI: 10.1080/17489725.2019.1682202

3 authors:

Oleksii Vedernikov
University of Melbourne
4 PUBLICATIONS **5** CITATIONS
SEE PROFILE

Lars Kulik
University of Melbourne
156 PUBLICATIONS **3,362** CITATIONS
SEE PROFILE

Kotagiri Ramamohanarao
University of Melbourne
512 PUBLICATIONS **9,699** CITATIONS
SEE PROFILE

Optimal Route Planning for Stochastic Time-Dependent Hitchhiker's Problem

Oleksii Vedernikov, Lars Kulik, Kotagiri Ramamohanarao
School of Computing and Information Systems
The University of Melbourne, Australia
{ovedernikov, lkulik, kotagiri} @unimelb.edu.au

February 24, 2020

Abstract

Hitchhiking is a travel mode characterized by unpredictable travel times involving several possible combinations of lifts on roads. In this paper, we formulate a hitchhiker's problem and develop a time-dependent stochastic route planning algorithm for hitchhikers. Namely, we introduce a concept of the stochastic time-dependent hitchhiking graph to find hitchhiking strategies with the least expected travel time or maximized reliability. We introduce various heuristics to prune the original hitchhiking graph to improve computational efficiency. We provide a complexity analysis of the problem and evaluate the proposed solution on real-world networks of several countries.

Keywords— Hitchhiking, ridesharing, route planning

1 Introduction

Hitchhiking is a well-known ridesharing transportation mode that is characterized by numerous hopping on and off random vehicles. Today, hitchhikers in Western countries constitute a large community of travellers with their subculture, clubs and social networks where they share their trip experience. For example, the website *Hitchlog* [15] contains trip records with a total length of more than 2,300,000 km, and another website *Hitchwiki* [16] contains annotations and users' ratings for 24,000 pick-up locations all around the world. Due to the unpredictability of hitchhiking, travel times might be significantly longer compared to those of private cars. Thus, an algorithm for providing a proper solution to handle multiple lifts, varying traffic and possible long waiting times is crucial in assisting trip planning for hitchhikers.

In a recent study [30], the concept of a *static hitchhiking graph* has been introduced to represent all possible lifts that a hitchhiker could take for travel. This model incorporates deterministic waiting times at pick-up locations and travel times on roads. However, it does not fully represent the unpredictable nature of hitchhiking as a transportation mode. The static hitchhiking graph does not include the variability of waiting and driving times during the day as well as their probabilistic nature. The waiting time for a hitchhiker at a certain location could vary more than travel time on roads for a car, so it follows a probabilistic distribution which varies over time of day due to the periodicity of road traffic flows.

In this paper, we tackle these challenges by introducing a *stochastic time-dependent hitchhiking graph*, which takes into account waiting and driving time distributions and their variations over time. Because of the variability of travel times, an optimal hitchhiking strategy may not include one shortest path but be comprised of several paths from some intermediate points depending on the arrival time. In contrast to most common shortest path algorithms which try to find an exact path to be traveled, we consider a *path program*, which specifies time-dependent decisions that should be taken by a hitchhiker depending on the arrival time at a location. We are finding the optimal hitchhiking path program to minimize the expected travel time and maximized route reliability.

Experiments show that computing optimal path programs might be computationally expensive, therefore heuristics to reduce the size of hitchhiking graphs without losing optimality for hitchhiking path programs are needed. We propose two methods for graph reduction based on the K-shortest path and penalty method with Customizable Contraction Hierarchies. The latter is shown to perform queries that are faster in orders of magnitude without sacrificing optimality. Our proposed solution is the foundation of a route recommender system for hitchhikers that can be extended to a ridesharing application with prearranged rides on a long-distance scale.

The main research question of this work is: for a given road network, time-dependent stochastic travel time distributions, start and destination points for a hitchhiker, how can we efficiently find a path program to minimize least expected travel time for hitchhikers? To answer this question, we analyze various hitchhiking hypergraph pruning techniques to reduce the computational time for real-world scenarios, utilizing the road networks and simulated traffic data from several countries.

The key contributions of this paper are:

- Formalization of stochastic time-dependent hitchhiking graphs to handle uncertain and varying waiting times for hitchhikers;
- Development of a framework to find the minimized expected travel time and most reliable hitchhiking path programs;
- Creation of efficient heuristics for reducing problem size without losing accuracy in path program recommendation.

The remainder of the paper is organized as follows. Section 2 provides a review of existing research about hitchhiking, stochastic time-dependent route planning and alternative graph construction. We will describe hitchhiker's problem and outline our solution in Section 3. Then, Section 4 contains the results of experiments for hitchhiking graphs for various countries. Lastly, Section 5 concludes our findings and describes future work for the hitchhiker's problem.

2 Literature review

2.1 Hitchhiking

To the best of our knowledge, the only paper about route planning for hitchhikers is a recent study be Vedernikov et al. [30], and the limitations of this paper regarding unrealistic and oversimplified travel times are discussed later. Another study [29] investigates and predicts waiting time and rating of pick-up locations for hitchhiking based on various features such as road type and distance to facilities. The paper does not address an issue of finding an optimal hitchhiking strategy. Previous research about various factors as gender or appearance, which determine the willingness of drivers to stop are summarized in a meta-analysis by Kotz [20].

2.2 Stochastic time-dependent route planning

First approaches of time-dependent deterministic route planning were made by Dreyfus [11] and Cooke and Halsey [7]. Bast et al [5] provide a comprehensive survey of different approaches for route planning, including time-dependent. Delling and Wagner [9] consider various speed-up techniques for time-dependent road network route planning, but they assume deterministic travel times. Hall [14] formulated the minimum expected time problem for stochastic time-dependent (STD) networks. He also showed that standard directed graphs cannot handle a problem of both time-dependent and stochastic route planning where routing directions might vary for different times. This work is the first one to introduce path program (also known as time-adaptive strategy and optimal routing policy) instead of the shortest path. Orda and Rom [26] provide a survey of different waiting conditions on a deterministic network. Pretolani [28] was the first to connect STD path program to the shortest hyperpath on a hypergraph, while a similar approach was used earlier by Nguyen and Pallotino [23] for urban transit applications. A recent approach by Bast and Storandt [6] utilizes the use of an expected frequency of transport instead of schedules. Miller-Hooks and Mahmassani [22] provide two similar algorithms - SDOT and ALET - to find least expected travel time path programs in STD networks, and ALET performs better for dense networks. Nielsen [25] extends hypergraph algorithms introduced by Gallo et al. [12] this idea to find the K best path programs and bicriterion path program in STD networks. Among other applications, time-dependent route planning has been proposed for flight networks [8]. However, for a hitchhiker's problem, the number of edges is much larger, and some of these methods become computationally infeasible. Also, most of the algorithms for optimal path programs on STD networks have been tested on simulated grid networks, not on real-world road networks.

2.3 Alternative graph construction

The most common application of alternative graphs is to provide alternative routing directions while navigation [4], and therefore its quality measures are usually related to the easiness of navigation while preserving optimality of produced paths. A related idea of alternative paths [3] finds the paths which share not more than a certain fraction of length with the shortest path, not too longer than the shortest path, and each segment of them is the shortest path itself.

K-shortest path algorithm to find top-K loopless shortest paths is a well-studied problem [31] that could be directly applied to find alternative graphs as a union of K-shortest paths. The Plateau method was introduced in [21], and its idea lies first in expanding two search trees: a forward tree from the source node and a backward tree from the destination node until they meet. Then, selecting the edges that appear in both searches (plateau) give a skeleton of an alternative graph.

The Penalty method is a state-of-the-art technique which iteratively computes shortest paths and increases their weights [27]. In [18] authors use the Customizable Route Planning technique for the quick computation of updating edge weights, which was later outperformed by Customizable Contraction Hierarchies (CCH) [10, 13]. To the best of our knowledge, we are the first to incorporate the CCH algorithm for the penalty method.

A related idea of reducing graph pruning without sacrificing the quality of path programs is considered in [19], but they consider different stochastic on-time arrival problem (SOTA) [24]. In order to reduce the complete road graph, they use Penalty, Plateau and Turn corridor methods. Turn corridor is computed as a union of all edges that could be taken with k detours from the shortest path. In the experiments, Turn and Penalty method provide a reasonable trade-off between sacrificing the most optimal strategy for the running time, while Plateau method is outperformed by Penalty method.

Also, all proposed alternative graph ideas are mostly related to the navigation, therefore they use

different criteria for assessing the quality of alternative graphs.

3 Problem description

3.1 Static/Stochastic Time-Dependent Hitchhiker's Problems

We begin by introducing hitchhiking as a transportation mode. In this scenario, a hitchhiker is willing to travel from a source city to a destination city on a certain road network. We assume that the road network is a country's highway system. In order to travel, hitchhikers must ask passing drivers for lifts. Drivers do not change their route but can stop at any node and a hitchhiker may leave the vehicle and wait for another lift at that point. Therefore, hitchhikers may have multiple lifts from different drivers in their travel. The hitchhikers can look for lifts to any destination node (e.g. writing the desired destination on a sign), not necessarily the next city on the current edge or their desired final destination. They can start and finish rides on all nodes of the road network, and can also wait for the lifts to the same destination at different roads starting within the same city.

The maximum number of possible travel options for a vehicle driver while driving is upper-bounded by the number of edges in a road network: $O(N)$, when all edges intersect in a single node. Unlike a vehicle driver, a hitchhiker can seek a lift at any road to any another town on the road network. Therefore, the total number of these desired lifts, or possible options a hitchhiker can take, is $O(N^2)$ as has been shown in [30], which comprises a complete graph. In both cases, it is assumed that the paths are loop-less. Therefore, routing for hitchhikers has significantly higher complexity than route planning for vehicle drivers.

In the model, time-independent and deterministic waiting and driving times as respective average values are used. The concept of a *static hitchhiker's problem* given as: for a given source and destination points on a certain road network and average travel times for hitchhikers on it, how to find the path with the shortest travel time? However, in the real world, waiting times for hitchhikers can vary significantly over time of a day. While driving on the same road takes almost the same time during day or night with considerably different delays due to congestion, waiting times might vary in hours due to the variability of traffic flows. Also, waiting times in the real world is never deterministic and is very probabilistic by nature depending on various factors. Therefore, their model is rather restricted, and our goal is to incorporate time-dependent and stochastic times and define it as a *stochastic time-dependent hitchhiker's problem*.

We incorporate the following properties into our country-scale hitchhiking model:

- All rides start and stop only at the road network nodes
- Hitchhikers can choose the next road to thumb up
- Hitchhikers can choose a desired next lift destination
- There are various combinations of lifts on a path
- Each road has a predefined best location to hitchhike which is characterized by a fraction of cars q_i which stop there
- Waiting time depends only on q_i and traffic on that road at a certain time
- Drivers' speeds are uniform
- Drivers follow the shortest paths on a road network
- Drivers are willing to stop independently of a hitchhiker
- A driver who stops can always take a hitchhiker on board

- There is the possibility to be dropped off before the driver's destination
- Travel time consists of waiting and driving time
- Travel time at an edge at a given departure time is a discrete random variable with a certain probability distribution.

In the following subsection, we will show how the difference between static and dynamic hitchhiker's problems will imply solving a significantly more difficult routing problem for the latter.

3.2 Static and Stochastic Time-Dependent Hitchhiking Graphs

First, we begin by introducing different types of graphs used in this study. They are shown in Figure 1.

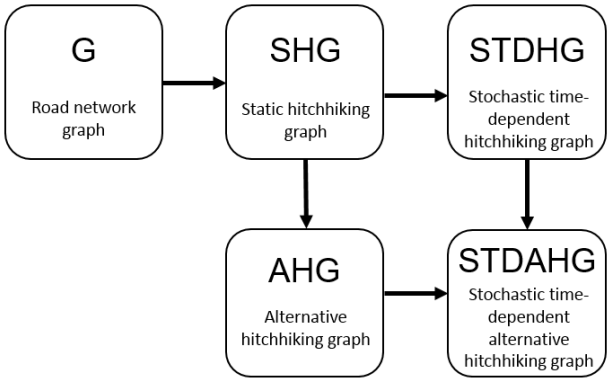

Figure 1: Schema of graphs

Road network graph $G = (N, E, l)$ is a weighted directed graph where N is a set of nodes (vertices) corresponding to cities and E set of edges corresponding to roads on a road network. $\forall e \in E$ its weight $l(e) : E \to \mathbb{R}$ is defined as a length of the corresponding road. An example of the road network graph is given in Fig. 2.

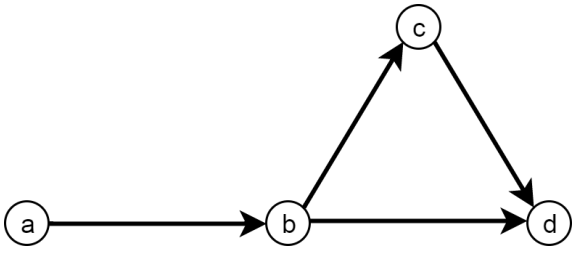

Figure 2: Road network graph G

The shortest path problem on G equal to finding a path with the minimum sum of weights on roads. This is a well-studied problem that generalizes the optimal route finding for vehicles.

However, the road network graphs G are not applicable to the problem of optimal route planning for hitchhikers since a hitchhiker can choose an intended destination at any intermediate node, not only the next edge. To handle this issue, the concept of static hitchhiking graph has been introduced [30]. Static hitchhiking graph $SHG = (N, HE, w)$ is a weighted directed graph where N is the same set of

nodes as G, while $HE = \{(n_i, n_j) | n_i \text{ can reach } n_j : n_i, n_j \in N\}$ is a set of hitchhiking edges, which correspond to all possible decisions that could be made by a hitchhiker traveling on a road network G: they can ask for a lift at any pick-up location to any other node from N. In order to annotate all hitchhiking edges $\forall e \in HE$ with corresponding travel times, weights $w(e) : HE \to \mathbb{R}$ are introduced which are equal to average travel time on the edge e. $w(e) = t(e) + \tau(e)$ is a sum of average waiting time $t(e)$ for a hitchhiker to be picked up and average driving time $\tau(e)$ on it. An example of SHG with weights $w(e)$ is shown in Figure 3. To construct these weights, new values are needed as traffic on roads and pick-up probabilities of locations on every road $e \in E$. We discuss how to obtain these parameters in Section 4. We also refer to hitchhiking edges as simply edges in this paper. Note the shortest path between a and d is 7.5.

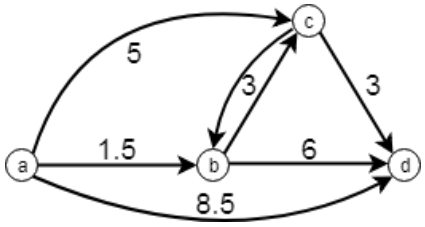

Figure 3: Static hitchhiking graph SHG with static travel times

To add stochastic and time-dependent components to SHG, we define *stochastic time-dependent hitchhiking graph* $STDHG$. Assume we have a discrete time horizon $T_{horizon} = \{0, 1, 2, \ldots, t_{max}\}$ corresponding to all possible timestamps in our problem. We define $STDHG = (N, HE, W)$ using N and HE the same as in SHG, while instead of average travel time we use a discrete random variable $w_{e,t} \in W$ with corresponding probability mass function (PMF) $f_{w_{e,t}}(x) = Pr[w_{e,t} = x]$, where its domain $\mathcal{D}(f_{w_{e,t}})$ is a set of all possible travel times and range $\mathcal{R}(f_{w_{e,t}})$ contains their probabilities. Therefore, a travel time along hitchhiking edge $e \in HE$ for each starting time t can have multiple values with corresponding probabilities from PMF $f_{w_{e,t}}(x)$, and let p_{max} be the maximum possible number of values among PMFs for $\forall e \in HE$. Therefore, stochastic and time-dependent travel times $w_{e,t}$ in $STDHG$ allow us to represent the unpredictable nature of hitchhiking and overcome overly simplistic approach of average travel times in SHG. A structure of $STDHG$ is the same as SHG and is shown in Fig. 3, while travel time distributions are shown in Table 1. Note that static travel times in Figure 3 are essentialy the weights $w(e)$ of SHG.

Table 1: Travel times of edges in SHG

(u,v),t	(a,b), 0	(a,c), 0	(a,d), 0	(b,c)$\forall t$	(b,d), 1	(b,d), 2	(c,d)$\forall t$
$\mathcal{D}(f_{w_{(u,v),t}})$	{1,2}	{5}	{8,9}	{3}	{4}	{8}	{3}
$\mathcal{R}(f_{w_{(u,v),t}})$	{0.5,0.5}	{1}	{0.5,0.5}	{1}	{1}	{1}	{1}

However, in such case, the notion of a shortest path is no longer applicable. For example, taking a ride along hitchhiking edge $e = (n_i, n_j)$ at time t_0, the travel time $w_{e,t} = w_{(n_i,n_j),t}$ might take multiple values from range $\{t_1, t_2, \ldots t_p\}$ according to PMF $f_{w_{e,t}}(x)$. However, the next hitchhiking edge from n_f in an optimal route might vary depending on different arrival times $\{t_1, t_2, \ldots t_p\}$ to n_f. Therefore, instead of a path, we are interested in a *path program* which is a rule that assigns the next road depending on both the current node and arrival time. We consider *a priori* scenario when a hitchhiker has to receive the whole strategy before the trip started due to the possible absence of network upon arrival at intermediate nodes.

Table 2: Expected travel times for TPP on $STDHG$

No.	path program $P_{a,d,0}$	$E(w_P)$
1	(a,b),(b,c),(c,d)	7.5
2	(a,b),(b,d)	7.5
3	(a,c),(c,d)	8
4	(a,d)	8.5
5	(a,b),[1:(b,c),(c,d),2:(b,d)]	8.5
6	(a,b),[1:(b,d),2:(b,c),(c,d)]	6.5

Definition 1 *For a given source and destination nodes $\{s,d\} \subset N$ and departure time $t_s \in T_{horizon}$, Travel Path Program (TPP) is a function $P_{s,d,t_s} : S \to HE$, where $S \subset \{(n,t) : n \in N, t \in T_{horizon}\}$ assigns for each intermediate node n and arrival time t the next hitchhiking edge $e \in HE$ that a hitchhiker has to take and satisfies following properties:*

1. $(s,t_s) \in S \wedge (s,t') \notin S \, \forall t' \neq t_s$
2. $\exists n,t : P_{s,d,t_s}(n,t) = (n,d)$
3. $\forall (n,t) \in S, t' \in \mathcal{R}(w_{(n,P_{s,d,t_s}(n,t)),t}) \implies (P_{s,d,t_s}((n,t),t+t') \in S$

An informal example of TPP of graph in Figure 3 and travel times from Table 1 from a to d at $t_s = 0$ will be: take an edge (a,b), then if arrive at $t = 1$, take (b,c) and then (c,d), otherwise take (b,d).

TPP P_{s,d,t_s}, also called a hyperpath, represents a hitchhiking strategy and provides routing choices for travelers from all nodes and leaving times in the time horizon. That is, a hitchhiker leaving node n at time t travels along hitchhiking edge $P_{s,d,t_s}(n,t)$. For example, if a traveler arrives at node n at time $10am$, they should follow a hitchhiking edge e_1, if they arrive at $11am$, they should travel along e_2 etc.

Due to probabilistic nature of hitchhiking edges in $STDHG$, for each TPP P_{s,d,t_s} the total travel time will be a discrete random variable $w_{P_{s,d,t_s}}$ with PMF $f_{w_{P_{s,d,t_s}}}(x)$. Therefore, the notion of the shortest path in SHG between s and d as a path with a minimum sum of travel times is no longer valid, so we can define Optimal Path Program as TPP with the least expected travel time $E(w_{P_{s,d,t_s}})$.

Definition 2 *For a given source and destination nodes $\{s,d\} \subset N$ and departure time $t_s \in T_{horizon}$, Optimal Path Program (OPP) P_{opt} is TPP between s and d such that: $E(w_{P_{opt}}) \leq E(w_P) \forall P_{s,d,t_s}$ that satisfies Definition 2.*

Table 2 has expected travel times for TPP on $STDHG$ from a to d and with $t_s = 0$. Path program 6 has the least expected travel time $E(w_P) = 6.5$, so it is OPP. Note that it is less than expected travel time on SHG which is 7.5.

Therefore, OPP is a hyperpath starting from s at time t_s that provides routing choices for all possible intermediate nodes and times, ends at d and has the least expected travel time. However, since this TPP might have travel times 5 or 8, in some cases TPP 1 or 2 might be preferred because they always have stable travel time and thus are more reliable. In Subsection 3.3.2, we discuss that instead of minimizing $E(w_P)$, a path reliability function can also be chosen.

3.3 Optimal Path Program for $STDHG$

Our notion of stochastic time-dependent hitchhiking graphs aligns with the previous research of stochastic time-dependent routing in different transportation networks. While the overall discussion about

literature findings regarding OPP is in Section 2, we summarize the findings of two algorithms to find OPP in $STDHG$.

3.3.1 ALET

Adaptive Least Expected Time (ALET) algorithm was proposed by Miller-Hooks [22] and is an algorithm to find OPP. For each node, it maintains a vector with the least expected times to the destination node known thus far, and this vector is updated while iterating over edges. The pseudo-code is given in Algorithm 1, assuming the departure time $t = 0$.

Algorithm 1 ALET algorithm

procedure ALET($STDHG, s, d$) ▷ Find OPP
  $Exp(n,t) \leftarrow \infty, n \neq d, t \in T_{horizon}$ ▷ Expected optimal
  $Exp(d,t) \leftarrow 0, t \in T_{horizon}$ ▷ travel time to d
  $Prev(n,t) \leftarrow None, n \in N, t \in T_{horizon}$ ▷ Prev. node in PP
  $Push(SE, d)$
  while $SE \neq \emptyset$ **do**
    $i \leftarrow Pop(SE)$
    for $j \in STDHG^{-1}(i)$ **do** ▷ $(j,i) \in HE$
      for $t \in T_{horizon}$ **do**
        $temp \leftarrow 0$
        for $t' \in R(w_{(j,i),t})$ **do**
          $sum \leftarrow sum + f_{w_{(j,i),t}}(t') \cdot (t' + Exp(i, t + t'))$
        end for
        if $temp < Exp(n,t)$ **then**
          $Exp(j,t) \leftarrow temp$
          $Prev(j,t) \leftarrow i$
          if $j \notin < SE$ **then**
            $Push(SE, j)$
          end if
        end if
      end for
    end for
  end while
  return $Exp(s), Prev(s)$
end procedure

Its worst-case complexity is $O(N^3 \cdot t_{max}^2 \cdot p_{max})$, where p_{max} is the maximum possible number of travel time probabilities, and t_{max} is a number of timestamps in $T_{horizon}$. Experiments presented in [22] show the actual running time is smaller than the worst-case scenario.

3.3.2 MinHypertree algorithm

Pretolani [28] proposed to use hypergraphs, where each edge can have multiple tail nodes representing different possible arrival times and connected OPP problem to finding a minimum weight hyperpath on a new hypergraph described below. Note we simplify some notations here comparing to the original paper.

Definition 3 *A weighted directed hypergraph $H = (V, A, W_H)$ is consists of a set of vertices V, hyperarcs A and weights $W_H : A \rightarrow \mathbb{R}$. A hyperarc $a = (Tail(a), head(a)) \forall a \in A$ where $Tail(a) \subset V$ is a set of tail vertices and $head(a) \in A \setminus Tail(a)$ is a head vertex.*

In order to construct $H = (V, A, W_H)$ from $STDHG = (N, HE, W)$: $V = \{(n, t)\ \forall n \in N, t \in T_{horizon}\} \cup \{s'\}$, $A = \{(\{(n_j, t') : t' \in \mathcal{R}(w_{(n_i,n_j),t})\}, (n_i, t))\ \forall (n_i, n_j) \in HE, t \in T_{horizon}\} \cup \{(\{s'\}, (d, t)) \forall t \in T_{horizon}\}$. Therefore, we construct a new vertex $v \in V$ for each pair of (node in N, timestamp from $T_{horizon}$) plus a new source s', and they are connected with hyperarcs $a \in A$ where a set of tail vertices corresponds to all possible probabilities of travel times from one node to another along a hitchhiking edge $e \in HE$. Note that its orientation is reversed. Besides, we connect a new source s' to all vertices which correspond to pairs (the original destination d, all possible arrival times t to d). The new destination $d' = (s, 0)$ is a vertex corresponding to the original source and departure time 0.

An example of hypergraph H constructed from example SHG from 3 and travel time distributions from Table 1 is shown in Figure 4, where for the simplicity nodes (n, t) are depicted as n_t. In this case, $d' = (a, 0)$, and s' is has hyperarcs to all (d, t) that correspond to possible arrival times to the original destination d.

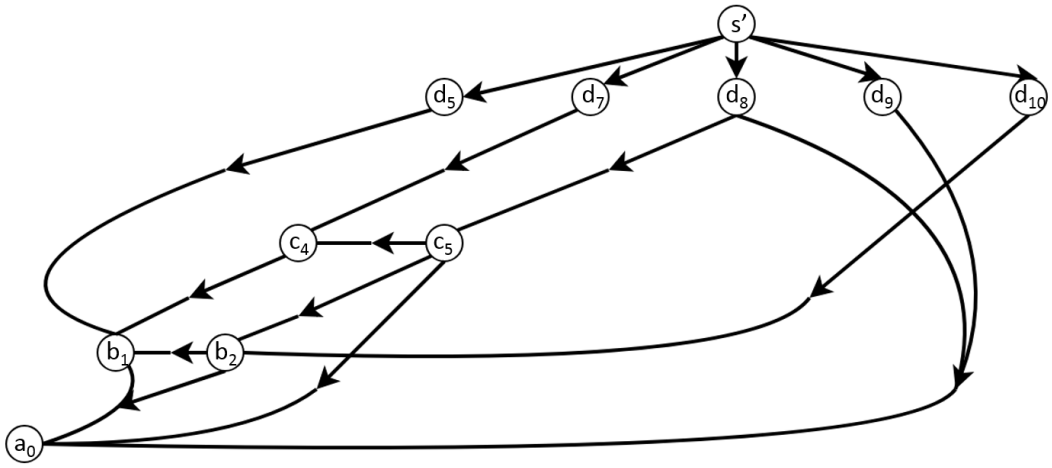

Figure 4: Hypergraph H constructed from hitchhiking graph SHG

For example, multiple arriving times $8, 9$ to d in a hitchhiking edge (a, d) are represented as a hyperarc with multiple tail vertices that correspond to them: head is a_0, and tail contains d_8 and d_9. This representation allows us consider multiple arriving times, and we introduce computation of a hyperpath weight that corresponds to the expected travel time of a certain TPP. A hyperpath between $s' - d'$ is defined using Definition 2. For any $s' - d'$ hyperpath its weight (or expected travel time on it) is calculated as

$$W_H(v) = \begin{cases} 0 & v = s' \\ w_H(p(v)) + \sum_{u \in Tail(p(v))} f_{w_{u,t}} W_H(u) & v \neq s' \end{cases}$$

where

$$w_H(a) = \begin{cases} t & a = (\{s'\}, (d, t)) \\ 0 & else \end{cases}$$

where p is a predecessor function which assigns a preceding arc a for each vertex v on a hyperpath. The weight of a hyperpath is related to all possible probabilities of travel times in all edges e and arrival times to d. In case of H, we need to find a minimum weight hyperpath between s' and d'. Note that it is also reversed to the original hyperpath between s and d as shown in 4. Using the assumptions that travel times are positive, vertices of a hypergraph H can be ordered $(s' = v_1, v_2, \ldots, v_{\|V\|})$ and we can

use the Algorithm 2 to find a minimum weight hyperpath.

Algorithm 2 Minimum weight hyperpath algorithm

```
1: procedure MinHypertree(H, s', d', V)
2:     Exp(v_i) ← ∞, ∀i ∈ {2, 3, ..., ||V||}
3:     Exp(s') ← 0
4:     for i = 2 to ||V|| do
5:         for a ∈ H^{-1}(v_i) do                    ▷ a = (Tail(a), v_i) ∈ A
6:             if W_H(v_i) > w_H(a)+
7:                 Σ_{u∈Tail(p(v))} f_{w_{u,t}} W_H(u) then
8:                 W_H(v_i) = w_H(a) + Σ_{u∈Tail(p(v))} f_{w_{u,t}} W_H(u)
9:         end for
10:    end for
11:    return Exp(d')
12: end procedure
```

The overall complexity of this algorithm is $O(\|HE\| \cdot t_{max} \cdot p_{max}) = O(N^2 \cdot t_{max} \cdot p_{max})$. With a minor change in the algorithm, $W_H(v_i)$ contains maximum possible travel time, and no calculation of expected travel time is needed. We analyse the usage of both algorithms in Section 4.5 and describe the difference between various weight functions in Section 4.7.

3.4 Alternative hitchhiking graphs

While solving the OPP on $STDHG$ will give the path program with the minimum expected travel time, finding it requires to include time-dependent and stochastic distributions for all edges in the graph. Each edge on SHG will result in multiple edges on $STDHG$ for each time interval which might result for different time programs and requires a lot of computational resources. Later, we show that computing ORP on a complete SHG will require several minutes even for a road network G with 200-300 nodes.

However, not all edges on SHG will be used for OPP on $STDHG$. This OPP will contain a set of edges which are close (i.e. adjacent or parallel) to the shortest path edges on SHG. While certain sub-optimal paths on SHG might have less travel time than the optimal path, considering the that travel times on both paths vary over time, they still could be a part of OPP for some time periods. Therefore, corresponding edges on SHG will be used in OPP for $STDHG$ in addition to the edges from the shortest path on SHG. At the same time, edges on SHG which lie far from the shortest path are less likely to be included in the OPP.

Therefore, we are interested in preselecting the edges from the original SHG which are more likely to be included in the $STDHG$. These edges constitute a new subgraph of SHG, and we define it as Alternative Hitchhiking Graph (AHG). Alternative graphs as a set of alternative paths to the shortest path. Their usage is discussed in Section 2.

After preselecting AHG from SHG, we can extend it to Stochastic Time-Dependent Alternative Hitchhiking Graph, or $STDAHG$ and find the OPP on it. Some of the edges in SHG might be a part of OPP on $STDHG$ but not be selected in AHG, therefore they will not appear in OPP on $STDAHG$. As a result, this OPP on a subgraph $STDAHG$ will have a larger expected time than OPP on a complete graph $STDHG$. Therefore, a quality measure of a method to construct AHG is a minimized difference between OPP on these graphs. Another important measure is how the reduced size minimizes query running time, and both of them will be discussed in Section 4. We introduce 2 methods of constructing AHG.

3.4.1 K-shortest paths

K-shortest path problem is a well-known generalization of the shortest path problem. It aims to find not only the shortest path on a graph, but also K shortest, or suboptimal, paths, in increasing order. Therefore, for each K, we can define AHG as a union of all edges which are included in the first K shortest paths. The larger K, the more edges are in the corresponding AHG. Also, new suboptimal paths are likely to contain combinations of previously included edges and do not produce new edges. We provide a listing of Yen's algorithm [31] in Algorithm 3.

Algorithm 3 K-shortest path

```
1: procedure KSP(G, s, d, K)
2:     A[0] ← BidirectionalDijkstra(G, s, d)
3:     B ← PriorityQueue()
4:     G_0 ← G
5:     for k = 1 to K − 1 do
6:         for i = 0 to length(A[k − 1]) − 2 do
7:             spurNode ← A[k − 1][i]
8:             rootPath ← A[k − 1][0 .. i − 1]
9:             for path ∈ A do
10:                if path[0 .. i − 1] = rootPath then
11:                    remove (path[i], path[i + 1]) from G
12:                end if
13:            end for
14:            for rootPathNode ∈ rootPath \ {spurNode} do
15:                remove rootPathNode from G
16:            end for
17:            spurPath ← BidirectionalDijkstra(G, spurNode, d)
18:            B.put(rootPath + spurPath)
19:            G ← G_0
20:        end for
21:        A[k] ← pop(B)
22:    end for
23:    return ∪_{∀e∈p,∀p∈A}{e}
24: end procedure
```

In this algorithm, A is an array of K-shortest path sorted by their length, B is a priority queue of candidate shortest paths. On each iteration, a new path is added to B, which consists of $rootPath$ and $spurPath$. $rootPath$ follows the first nodes from one of A paths, and $spurPath$ is a shortest path on a graph with eliminated edges from paths from A. The resulting alternative graph is a total set of all edges from all K-shortest paths.

3.4.2 Penalty method

The Penalty method [27] is an iterative run of the shortest path algorithm while increasing the weights of each shortest path while iterating. They are multiplied by W_{update}, and the total number of iterations is N_{iter}. In this case, the new edges are included in the shortest paths, and thus in the resulting alternative graph. This method requires a large number of shortest path runs with graph weight updates. We use CCH method [10], which for a given graph G and node order $\pi : \{1 \dots \|HE\|\} \to N$ constructs an upward directed graph $G^{\wedge}$ where all edges $(\pi(i), \pi(j))$ satisfy $i < j$, thus keeping edges which lead to nodes with larger order and contract other while preprocessing. Then, it performs an update procedure where edge weights are updated. For each query, bidirectional Dijkstra's algorithm is performed and

the resulting path is restored using preprocessed shortcuts. As discussed in 2.3, this work is the first to incorporate CCH in the Penalty method. We provide the listing of Penalty method using preprocessed graph $G^{\wedge}_{\pi}$, query and weight update of CCH speed-up technique to find shortest paths in Algorithm 4.

Algorithm 4 Penalty method algorithm

```
1: procedure PENALTY(G, s, d, π, N_iter)                    ▷
2:     A ← [ ]
3:     G^∧_π ← CCH_Preprocessing(G, π)
4:     for i = 0 to N_iter − 1 do
5:         p = CCH_Query(G^∧_π, s, d)
6:         A ← A ∪ {p}
7:         for (x, y) ∈ p do
8:             l_{x,y} ← l_{x,y} · W_update
9:         end for
10:        G^∧_π ← CCH_UpdateWeights(G^∧_π, l)
11:    end for
12:    return ∪_{∀e∈p,∀p∈A} {e}
13: end procedure
```

One of the most important parameters in its implementation is the order of contraction which corresponds to their relative importance in the graph. We propose to use the orders based on node population, degree centrality (number of adjacent edges), and betweenness centrality (fraction of shortest paths passing through a node).

3.5 Framework

Our framework is shown in Figure 5. All input data is shown in dotted boxes. The goal of the framework is to recommend a user OPP for their intended trip, and the user could choose either least expected travel time or most reliable path programs. The framework includes two main parts: the preprocessing and query stages. The result of the preprocessing stage is constructing SHG with all stochastic distributions of travel times $w_{e,t}$ stored in memory. We do not consider its constructing time because it does not affect query performance. To solve the cold start problem, we use simulated data for all required input data as described in Section 4, and then the preprocessing stage could be rerun using the newly acquired data from the users of the application.

The query stage models user's behaviour while using the recommender system. A user inputs source and destination locations and the initial time as their query. For each query, our framework finds OPP using either $STDHG$ or $STDAHG$, and we compare their performance in Section 4 as well as the performance of two methods of finding OPP.

4 Experiments

4.1 Experimental setup and parameters

For the settings of the experiments, we use road networks of European countries from DIVA-GIS [1] and population grids from Eurostat dataset [2] that are listed in the Table 3. An example of a road network of Germany is shown in Figure 6.

For each edge of the road network, assume q_i is a pick-up probability of the best pick-up location on it. The pick-up probabilities q_i are normally distributed $N(0.2, 0.1)$. We also assume the minimum

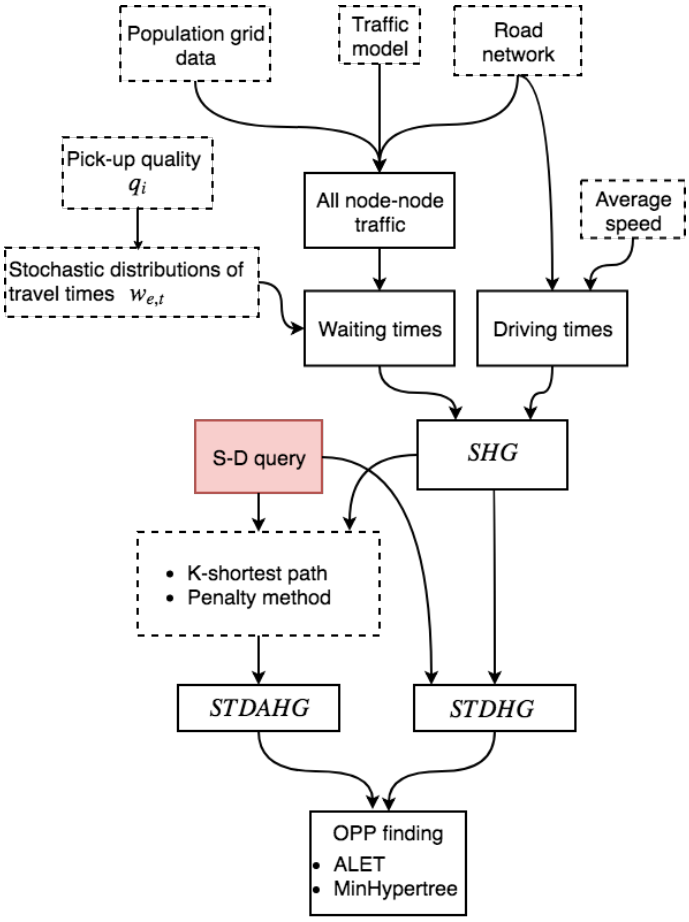

Figure 5: Framework

Table 3: Road networks and population grids used

Country name	**No. of nodes**	**No. of edges**	**Total population**	**Total road length, km**
Austria	468	1194	8425013	14946.09
Belgium	418	1312	10993324	10397.71
Czechia	407	1096	10447899	13967.01
France	3081	9532	62803124	122381.61
Germany	2533	7842	80232606	79935.83
Hungary	572	1496	9964691	18953.76
Italy	1715	5153	59485199	60975.23
Netherlands	380	1082	16655999	8936.33
Poland	1708	5196	38539668	70924.74
Portugal	385	1102	10577568	16811.03
Slovakia	270	692	5406725	9196.07
Spain	2060	6210	46975968	93652.32
Switzerland	340	914	7982181	9235.513

Figure 6: Road network in Germany

pick-up probability is equal to 0.001, i.e. in average 1 out of 1000 cars will stop in the worst case. The population of each grid is assigned to the closest vertice of the road network. Then, the traffic between each pair of nodes in the road network is estimated using the gravity law [17], and they all are supposed to take the shortest path between those nodes on a road network. The estimated number of vehicles per year is $traffic(i,j) = 0.00135 \cdot \frac{(P_i \cdot P_j)^{1.02}}{d_{ij}^2}$ where P_i, P_j are populations of corresponding cities and d_{ij} is a distance in km between them. To tune the constants of this gravity model, we use real-world hitchhiking waiting times for each country gathered from Hitchwiki dataset [16]. This traffic is assigned between all pairs of nodes on the shortest path. After that, a static hitchhiking graph SHG is created. Each road's traffic is assigned to all hitchhiking edges which pass through it. The travel time on a hitchhiking edge is a sum of waiting time on its first edge and driving time along the roads on it. The waiting time is calculated using q_i and *traffic(i,j)*, and the driving time is estimated using the average driving speed v_{dr} (equal to $90km/h$) and the length of the road corresponding to the edge. Therefore, the more the pick-up probability q_i is, the less waiting and thus total travel time is. We discretize all

travel times using a parameter $\delta = 15min$, so all times are discretized to $T_{horizon} = \{15, 30, 45, \dots 1440\}$. We measure travel times for hitchhikers in minutes. We use $p_{max} = 4$ as the number of various travel times in $\mathcal{D}(f_{w_{e,t}})$. We use Yen's algorithm for k-shortest paths, and k is in range $[10, 20, 30, 40, 50]$. For the Penalty method, we use $N_{iter} = [5, 25, 50, 100]$ and $W_{update} = 1.2$ as suggested in [27].

To find a running time of queries or hitchhiking travel time for a specific dataset and method, we use 100 source-destination queries. We simulate the usage of the hitchhiking route planning application, and select source-destination pairs using weighted random selection based on the population of corresponding nodes: the higher the population is, the higher the probability is. Since the number of edges on a full SHG is $O(N^2)$ and most of them have weights which are larger than shortest paths between corresponding nodes, we reduce those which cannot appear in OPP even with high travel time variability (more than 1.5 larger than the shortest path). We use Yen's algorithm for k-shortest paths, and k is in range $[10, 20, 30, 40, 50]$. For the Penalty method, we use $N_{iter} = [5, 25, 50, 100]$ and $W_{update} = 1.2$ as suggested in [27]. In some cases, we show only the representative results of some countries or queries. For the experiments, we use Python and C++ code and run experiments on Linux VM with 12Gb RAM.

4.2 Hitchhiking graphs for different countries

See Table 4 about hitchhiking graphs used in the experiments. We see that in countries with small distances and high population (i.e. the Netherlands), waiting times is slower and hitchhiking could be very fast with the least ride changes. However, countries with less population (i.e. Austria) could have considerably larger waiting times and might require more ride changes.

Table 4: Hitchhiking graphs

Country	**Number of nodes**	**Number of hitchhiking edges**	**Fraction of hitchhiking edges in reduced SHG**	**Median waiting time, min**	**Median driving time, min**	**Median total travel time, min**	**Fraction of waiting time to travel time**	**Median number of rides**
Austria	468	7291	0.033	128.59	416.6	545.19	0.309	7
Belgium	418	7705	0.044	2.79	29.37	32.16	0.095	5
Czechia	407	6686	0.040	25.340	157.42	182.76	0.16	6
France	3081	231637	0.024	16.1	478.01	494.1	0.034	12
Germany	2533	289086	0.045	36.92	323.27	360.19	0.114	9
Hungary	572	8208	0.025	49.83	164.02	213.85	0.304	7
Italy	1715	151528	0.052	32.01	82.17	114.18	0.390	3
Netherlands	380	11934	0.083	1.99	15.14	17.14	0.132	2
Poland	1708	72740	0.025	47.02	224.62	271.64	0.209	6
Portugal	385	7680	0.052	154.19	282.40	436.6	0.546	9
Slovakia	270	3187	0.044	49.64	98.25	147.89	0.505	3
Spain	2060	121427	0.029	8.24	222.12	230.35	0.037	2
Switzerland	340	5531	0.048	5.16	49.56	54.72	0.104	5

4.3 Performance of OPP on $STDAHG$ and $STDHG$ for small networks

We find OPP for both $STDHG$ and $STDAHG$ on small networks. so it is possible to assess the performance of algorithms to construct AHG. OPP on $STDHG$ provides the best possible hitchhiking strategy, while OPP on $STDAHG$ provides a suboptimal result if the edges included into OPP for SHG are not included in AHG while construction. Therefore, we can assess the optimality of AHG construction algorithms by calculating how much of the optimal expected hitchhiking travel time is preserved. The fraction of optimal expected travel times on $STDAHG$ and $STDHG$ is indicated in the column "Accuracy" in Table 5. Also, for each of the method, we compute the proportion of query running time in "Query speed-up" column. Penalty method gives almost perfect results for large N_{iter}, and saves a few minutes considering a sum of the query running time and travel time for the second graph. Even for the smallest road networks in our dataset, computing $STDHG$ takes too long, so we need to construct AHG first.

Table 5: Running times for $STDHG$ and $STDAHG$

Method, k **or** N_{iter}	**Number of edges in** AHG	$STDAHG$ **query running time, s**	**OPP on** $STDAHG$ **expected travel time, min**	**Query Speed-up**	**Accuracy, %**
$STDHG$ with 97 nodes and 1314 edges. $STDHG$ query running time 12.27 s; OPP expected travel time: 303.14 min					
K-s.p. 10	14.17	0.069	322.72	**177.27**	**93.94**
K-s.p. 30	23.76	0.185	312.80	**66.23**	**96.92**
K-s.p. 50	29.64	0.294	310.17	**41.68**	**97.73**
Pen. 5	14.75	0.016	310.49	**773.87**	**97.63**
Pen. 25	43.75	0.086	304.66	**142.87**	**99.50**
Pen. 50	67.19	0.154	303.25	**79.44**	**99.97**
$STDHG$ with 197 nodes and 5208 edges. $STDHG$ query running time 432.374 s; OPP expected travel time: 461.28 min					
K-s.p. 10	14.98	0.425	519.03	**1016.73**	**88.87**
K-s.p. 30	23.91	0.441	502.79	**980.72**	**91.74**
K-s.p. 50	29.47	0.452	497.02	**955.52**	**92.81**
Pen. 5	21.27	0.459	482.84	**941.74**	**95.54**
Pen. 25	68.05	0.762	464.71	**567.75**	**99.26**
Pen. 50	110.15	1.175	462.97	**367.86**	**99.64**

4.4 K-shortest path and penalty method parameters variation

For the analysis of how hitchhiking travel time and query running time variations over k for K-shortest path algorithm and N_{iter} in Penalty method, see Figure 7 and Figure 8. Both algorithms follow the linear time increase with increasing k and N_{iter}, while the gain in terms of hitchhiking travel time decreases.

4.5 Results for ALET vs MinHypertree

Comparative results for ALET and MinHypertree algorithms for AHG in various countries are shown in Table 6. We see that while ALET works faster for smaller graphs, it gets slower the larger graphs.

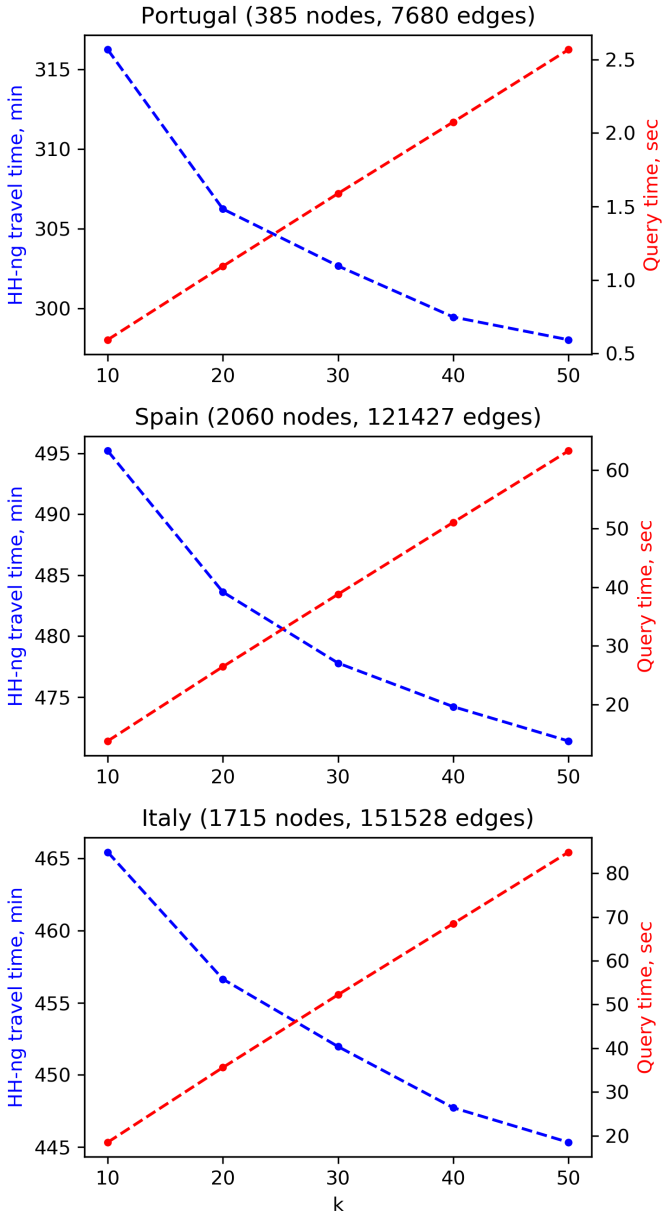

Figure 7: K-shortest path method running time

Even though MinHypertree requires more time for constructing all possible pairs of (node, timestamp), its better computational complexity provides better results in practice as well.

4.6 Results on different node orders for CCH

Since the insertion node order is one of the most important parts of CCH method, we investigate the following orders: random, total node population, degree centrality, betweenness centrality. The results are shown in Figure 9, and degree centrality gives the best results on average.

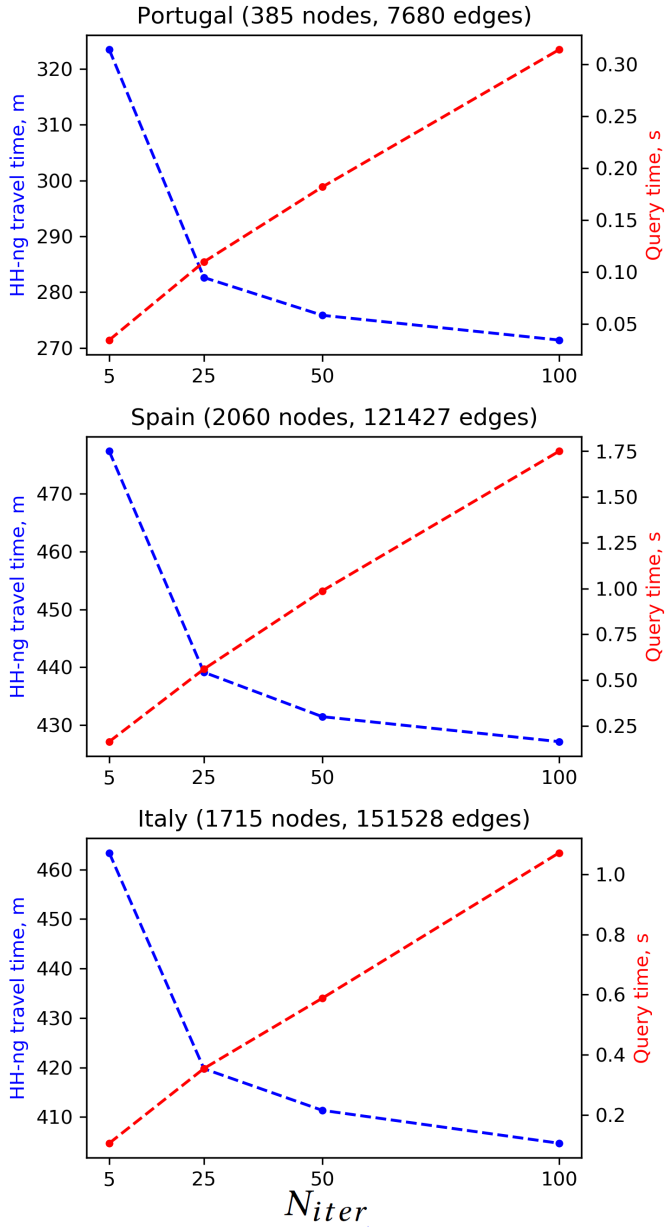

Figure 8: Penalty method running time

4.7 Results for expected travel time and reliability

The results of OPP with two objective functions using MinHypertree are shown in Table 7. Finding TPP with the minimized maximum travel time does not require computation of weighted probabilities to times, so only comparison and assignment of the maximum possible travel time are done. Therefore, computing the most reliable path program is done more than twice faster as shown in the "Fraction" column.

Table 6: ALET and MinHypertree running times

Country	**Edges in** AHG	**ALET running time, s**	**MinHypertree running time, s**	**Fraction**
Slovakia	11.39	0.013	0.053	**0.246**
Hungary	15.92	0.023	0.072	**0.318**
Netherlands	28.75	0.100	0.332	**0.300**
Austria	29.16	0.117	0.170	**0.691**
Switzerland	30.90	0.073	0.116	**0.626**
Poland	46.40	0.271	0.228	**1.187**
France	54.27	0.526	0.370	**1.420**
Italy	83.14	1.651	0.596	**2.771**
Germany	92.67	1.686	0.508	**3.321**

Table 7: Expected travel time vs. most reliable route

Country	**Median OPP for least expected travel time, ms**	**Median OPP for minimized maximum travel time, ms**	**Fraction**
Austria	19.001	9.032	**0.475**
Belgium	16.942	7.733	**0.456**
Czechia	8.741	4.164	**0.476**
Hungary	7.833	3.780	**0.483**
Netherlands	17.002	7.640	**0.449**
Portugal	17.739	8.191	**0.462**
Slovakia	8.980	4.293	**0.478**
Switzerland	16.735	7.713	**0.461**

4.8 Hitchhiking travel time vs query running time for large graphs

Figure 10 and Figure 11 show the performance of resulting algorithms to compute OPP using MinHypertree on $STDAHG$. The running time for Penalty method is significantly lower since it is using fast CCH technique to update weights and quickly find shortest paths. While K-shortest path has feasible running time for medium-sized networks, the query running time reaches a minute while the hitchhiking travel time of a found OPP is significantly less than the one found by Penalty method.

4.9 Discussion

Even though we use real-world hitchhiking waiting times from Hitchwiki dataset, the data they provide does not have a desired destination of a hitchhiker or trip time. While data like hourly road congestion on country-scale highway networks are not available, our simulated hitchhiking traffic model provides reasonable results for a proposed route recommendation application. Later, the gathered real-world data can be used to tune the parameters of our framework.

Computing $STDHG$ and $STDAHG$ on small graphs, we can compare performance of our AHG construction algorithms to the best-case scenario, and in most cases, the resulting AHG contains all edges from OPP on SHG, especially for Penalty method. Therefore, the resulting accuracy exceeds 99% for them, meaning optimal path programs on $STDAHG$ is almost equal to that one on $STDHG$, while the query speed is faster in orders of magnitude. We see that behavior of our AHG construction

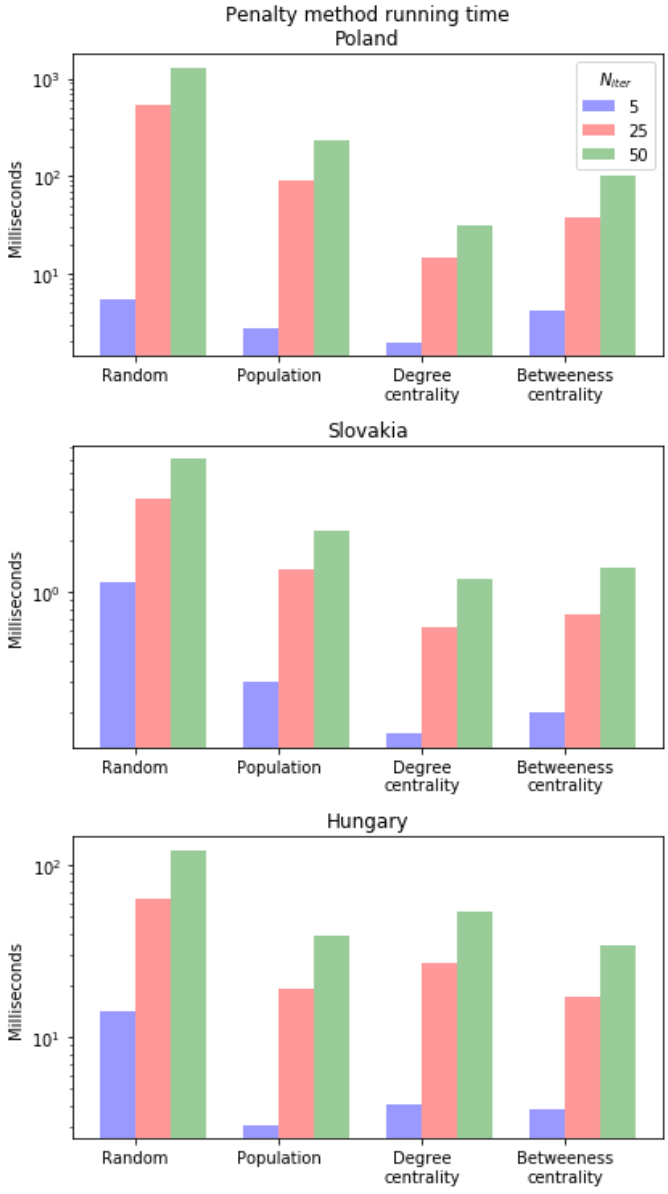

Figure 9: Penalty method running time vs CCH order

methods K-shortest path algorithm works reasonably well only for graphs with a few hundreds of nodes, while Penalty method with CCH performs queries within $1-2$ seconds for networks with a few thousand nodes and hundreds of thousands of hitchhiking edges. Regarding computing OPP for $STDHG$ and $STDAHG$, MinHypertree performs faster on larger networks. Regarding the node order for CCH, all three proposed methods provide significant query speed improvement comparing to the random ordering. Among these methods, degree centrality gives the best average results for all countries. Even though the preprocessing time for CCH can take a few minutes, it is outperformed by the fast query speed. Overall, implementing penalty method to compute AHG with MinHypertree to find OPP will allow to us to find an OPP with least expected travel time or most reliable travel time within 2 seconds

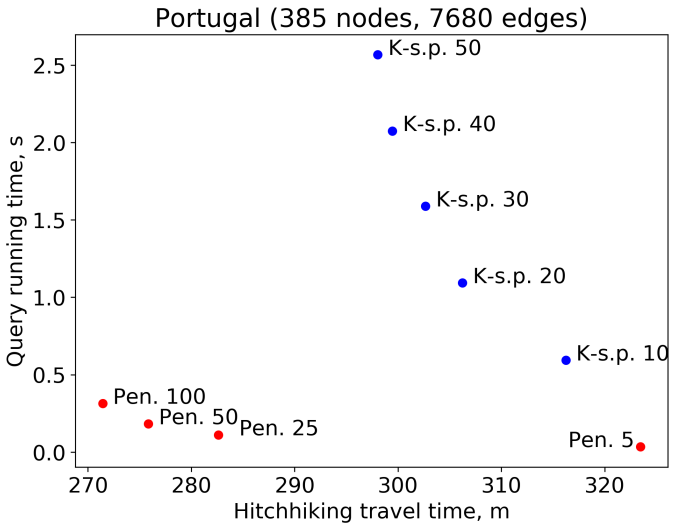

Figure 10: An example of a medium size $STDAHG$

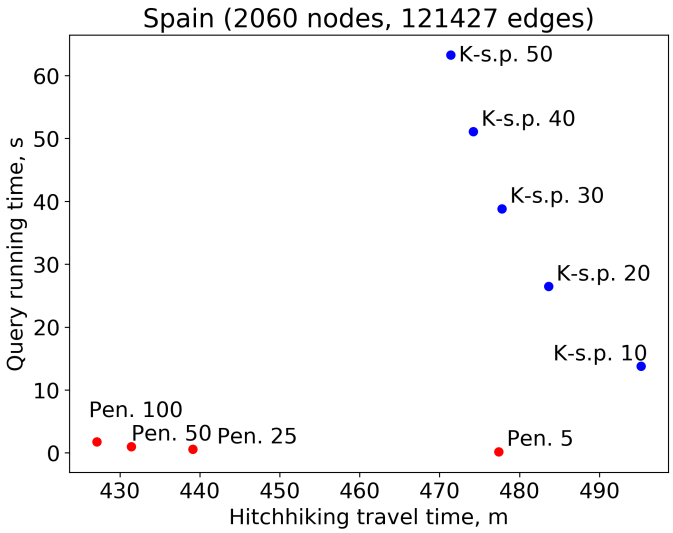

Figure 11: An example of a large size $STDAHG$

for largest available road networks.

5 Conclusions and future works

In this study, we formalize a problem of optimal route planning for the stochastic time-dependent hitchhiker's problem. We introduced a concept of the stochastic time-dependent hitchhiking graph to utilize all possible decisions for a hitchhiker and handle multiple uncertainties that are inevitable in hitchhiking as a transportation mode. We also applied several algorithms to reduce the size of the graph, heuristics to improve the computational speed and conducted a set of experiments on real-world

road networks of selected countries. We performed queries for finding optimal path programs of least expected travel time or most reliable travel time and showed that their query running time lies within $1-2$ seconds even for large road networks, while more than 99% optimality of routes is preserved. This framework is the basis of the future route recommender app for hitchhikers, which will allow optimal trip planning for hitchhikers on a country level.

Since this work is the first to tackle route planning for hitchhikers, a variety of future directions is possible. First, our framework could be extended into a long-distance ridesharing Uber-like application, where all prearranged rides will automatically change corresponding waiting times to zero and multiple rides would be allowed. A question of multiple pick-up locations might be considered. In this case, optimizing the sum of walking time from the initial location and waiting times might be of interest. More advanced heuristics for optimizing time-dependent hitchhiker's graphs might include insights from time-geography concepts, especially introducing time-space prisms to utilize all possible decisions for a hitchhiker within a certain time budget. A game-theoretical perspective of different hitchhikers and limited car capacity is another important direction. In this case, the more hitchhikers go to a location, the larger the waiting time becomes. Other problems of route planning could be considered, as using different notions of the reliability of path programs based on standard deviation or estimating departure for the desired arrival time.

6 Disclosure statement

No potential conflict of interest was reported by the authors.

References

[1] Diva-gis country data, 2017.

[2] Geostat 2011 grid dataset, 2017.

[3] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Alternative routes in road networks. *Journal of Experimental Algorithmics (JEA)*, 18:1–3, 2013.

[4] R. Bader, J. Dees, R. Geisberger, and P. Sanders. Alternative route graphs in road networks. In *Theory and Practice of Algorithms in (Computer) Systems*, pages 21–32. Springer, 2011.

[5] H. Bast, D. Delling, A. Goldberg, M. Mller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route Planning in Transportation Networks. *arXiv:1504.05140 [cs]*, Apr. 2015. 00117 arXiv: 1504.05140.

[6] H. Bast and S. Storandt. Frequency-based Search for Public Transit. In *Proceedings of the 22Nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '14, pages 13–22, New York, NY, USA, 2014. ACM. 00008.

[7] K. L. Cooke and E. Halsey. The shortest route through a network with time-dependent internodal transit times. *Journal of Mathematical Analysis and Applications*, 14(3):493 – 498, 1966.

[8] D. Delling, T. Pajor, D. Wagner, and C. Zaroliagis. Efficient Route Planning in Flight Networks. In *Proceedings of the 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'09)*, 2009.

[9] D. Delling and D. Wagner. Time-Dependent Route Planning. In R. K. Ahuja, R. H. Mhring, and C. D. Zaroliagis, editors, *Robust and Online Large-Scale Optimization*, number 5868 in Lecture

Notes in Computer Science, pages 207–230. Springer Berlin Heidelberg, 2009. DOI: 10.1007/978-3-642-05465-5_8.

[10] J. Dibbelt, B. Strasser, and D. Wagner. Customizable contraction hierarchies. In *International Symposium on Experimental Algorithms*, pages 271–282. Springer, 2014.

[11] S. E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations research*, 17(3):395–412, 1969.

[12] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2):177–201, Apr. 1993.

[13] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012.

[14] R. W. Hall. The fastest path through a network with random time-dependent travel times. *Transportation science*, 20(3):182–188, 1986.

[15] Hitchlog. Hitchhiking logs, 2017. [Accessed June 2017].

[16] HitchWiki. Hitchhiking maps and wiki, 2017. [Accessed June 2017].

[17] W.-S. Jung, F. Wang, and H. E. Stanley. Gravity model in the Korean highway. *EPL (Europhysics Letters)*, 81(4):48005, Feb. 2008. 00106 arXiv: 0710.1274.

[18] M. Kobitzsch, M. Radermacher, and D. Schieferdecker. Evolution and evaluation of the penalty method for alternative graphs. In *ATMOS-13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems-2013*, volume 33, pages 94–107. Schloss DagstuhlLeibniz-Zentrum fuer Informatik, 2013.

[19] M. Kobitzsch, S. Samaranayake, and D. Schieferdecker. Pruning techniques for the stochastic on-time arrival problem\texorpdfstring–an experimental study. *arXiv preprint arXiv:1407.8295*, 2014.

[20] F. Kotz. The base-rate of hitch-hiking success and its moderators: A meta-analysis. *Transportation Research Part F: Traffic Psychology and Behaviour*, 46, Part A:149–160, Apr. 2017. 00000.

[21] C. V. I. T. Ltd. Choice routing, 2009.

[22] E. D. Miller-Hooks and H. S. Mahmassani. Least Expected Time Paths in Stochastic, Time-Varying Transportation Networks. *Transportation Science*, 34(2):198–215, 2000.

[23] S. Nguyen and S. Pallottino. Equilibrium traffic assignment for large scale transit networks. *European Journal of Operational Research*, 37(2):176 – 186, 1988.

[24] Y. Nie and Y. Fan. Arriving-on-time problem: discrete algorithm that ensures convergence. *Transportation Research Record*, 1964(1):193–200, 2006.

[25] L. R. Nielsen et al. *Route choice in stochastic time-dependent networks*. University of Aarhus. Department of Operations Research, 2004.

[26] A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM)*, 37(3):607–625, 1990.

[27] A. Paraskevopoulos and C. Zaroliagis. Improved alternative route planning. In *OASIcs-OpenAccess Series in Informatics*, volume 33. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[28] D. Pretolani. A Directed Hypergraph Model for Random Time Dependent Shortest Paths. *European Journal of Operational Research*, 123, 1998.

[29] O. Vedernikov, L. Kulik, and K. Ramamohanarao. The hitchhiker's guide to the pick-up locations. *Open Geospatial Data, Software and Standards*, 1(1):12, Dec. 2016. 00000.

[30] O. Vedernikov, L. Kulik, and K. Ramamohanarao. The hitchhiker's guide to the optimal route planning. In *Proceedings of the 2017 IEEE 18th International Conference on Mobile Data Management*, MDM '17. IEEE Computer Society, 2017.

[31] J. Y. Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.