Using path decomposition enumeration to enhance route choice models

Peer-reviewed author version

KNAPEN, Luk; Hartman, Irith Ben-Arroyo & BELLEMANS, Tom (2020) Using path decomposition enumeration to enhance route choice models. In: FUTURE GENERATION COMPUTER SYSTEMS-THE INTERNATIONAL JOURNAL OF ESCIENCE, 107, p. 1077-1088..

DOI: 10.1016/j.future.2017.12.053
Handle: http://hdl.handle.net/1942/25602

Using path decomposition enumeration to enhance route choice models

Luk Knapen[a,*], Irith Ben-Arroyo Hartman[b], Tom Bellemans[a]

[a] *Hasselt University, Transportation Research Institute, Martelarenlaan 42 B3500 Hasselt, Belgium*
[b] *University of Haifa, Caesarea Rothschild Institute, Haifa 3498838, Israel*

Abstract

Prediction of realistic routes is essential in travel behaviour research that evaluates the effects of infrastructure design alternatives. Most proposed route choice models are based on additive link attributes. This paper investigates the decomposition of a given path in a graph into least cost components. This corresponds to finding the smallest number of intermediate destinations between which the traveler moved using the most efficient path. Minimum path decompositions are not unique and hence a single given path may result in multiple sets of intermediate destinations. This paper presents a technique to enumerate all possible decompositions of revealed paths and shows how to determine which road network nodes are preferentially used as intermediate destinations.

This paper explains the decomposition enumeration process and focuses on a new algorithm to enumerate efficiently all path decompositions. We implement the algorithms on 500k predicted bikers routes in Amsterdam.

Keywords: Graph Theory, Route Choice, Simulation, GPS Traces

1. Introduction

Traveler preferences can be extracted from routes recorded by GPS traces. Map-matching such traces leads to link sequences in a road network; these can

*Corresponding author
Email addresses: luk.knapen@uhasselt.be (Luk Knapen), irith.hartman@gmail.com (Irith Ben-Arroyo Hartman)

be analyzed using graph theoretical concepts. The research reported in this paper focuses on *utilitarian* trips i.e. movements executed in order to perform an activity at a given location (as opposed to fun trips that often consist of a closed *walk* in a graph). Hence, the trips considered in this paper are the ones that lead to *paths* in a graph.

Map matching GPS traces results in sequences of road network links crossed by the travellers. Paths extracted from collections of GPS traces provide revealed preference information about the individuals' route choice. This revealed preference is analyzed in order to find out how people construct their routes by concatenating least cost (shortest) paths. This is equivalent to choosing a sequence of intermediate destinations and moving between them using shortest paths. From this analysis one can find out whether particular nodes in the network appear to act as intermediate destinations more frequently than other ones. If sufficient traces are available for each traveler, it is possible to determine whether such attraction is caused by network properties or by personal preferences. As a consequence, the analysis results are relevant to simulation in mobility science. Whenever travel demand is known as a set of tuples $\langle startTime, mode, origin, destination \rangle$ the researcher is interested how the individuals will move in the network. Such simulations aim to quantify the effects of changes in the network infrastructure on travel patterns. Hence they support investment analysis.

In Knapen et al. [1] it was claimed that when a traveller considers a choice of route between an origin and a destination, he/she does not necessarily choose the quickest/fastest/cheapest route, but a route which is a concatenation of small number of shortest/cheapest routes. A decomposition of a path into least cost paths that has the smallest possible number of intermediate destinations is a *minimum path decomposition* and the minimum number of such least cost subpaths is called the *complexity* of the path. The distribution of path complexities, extracted from GPS traces [1], can be used to enhance the quality of route choice models.

In this paper, the terms *network*, *node*, *link* and *route* are used when trans-

portation concepts are discussed. The terms *graph*, *vertex*, *edge*, *path* and *walk* are used when discussing graph theoretical concepts.

2. Research Context

In this overview we discuss path based models, recursive link based models and research that focuses on a hierarchical method.

2.1. Path based methods

Path based methods define discrete choice models to select a path from an origin location L_O to a destination location L_D from a set of alternatives. The choice set needs to be established and the parameters for the choice model need to be estimated.

2.1.1. Basic concepts and comparative research

Bovy [2] presents a framework to describe the route choice problem. The author emphasizes the importance of both the choice set generation and the choice process per se. It presents excellent insight into the basic concepts of path based route choice.

Prato [3] presents an overview of the state of the art in route choice modeling by showing details about the available techniques for choice set formation and for the discrete choice process.

Prato [4] compares several techniques using numerical experiments. Subjective choice sets are generated using 6 path generation techniques (k-shortest paths, link penalties, branch-and-bound, stochastic link impedance, doubly stochastic model and random walk) and 5 parameter sets for each generator (hence 30 cases). Then `PSC` Path Size Correction logit (Bovy) is postulated to be used in reality and it is applied to all subjective sets using `length`, `speedBumps`, `leftTurns` and `PSC`: this is used to create synthetic data (postulated subjective selected routes). In a second stage objective choice sets are generated from the synthetic data using the same parameters and techniques. This results in

30[subj]*30[obj]=900 cases which are used to compare the postulated selected routes with the predicted ones.

2.1.2. Particular techniques

Prato and Bekhor [5] present a branch and bound based algorithm to generate the route choice set. Routes are selected using the deterministic shortest path criterion in a network with randomized link properties. Generated routes are evaluated using several constraints (number of left turns, travel time, etc) to decide about their addition to the choice set.

Prato and Bekhor [6] focuses on the effects of choice set composition. Different generation techniques are compared. Prediction accuracy is investigated and the branch and bound technique is found to outperform other ones. The paper concludes that parameter estimation is more robust for non-nested choice models (C-logit and path size logit).

Frejinger [7] proposes a stochastic path generation algorithm to populate the choice set by using a random walk generator starting at the origin and constrained to head towards the destination. The paper provides a sampling technique based on the Path Size attribute (Commonality Factor) that takes into account the correlation structure caused by overlapping paths. It is argued that the Path Size attribute needs to be evaluated over the set of all paths, which is infeasible, and the paper determines a minimum set required to obtain unbiased results. The technique was applied to a synthetic network and for real networks it is reported by Rieser-Schüssler et al. [8] and Vacca et al. [9] to be slow. Rieser-Schüssler et al. [8] also reports it to be very sensitive to selected input parameters that severely affect the distribution for the lengths ratio *actual/shortest* of the generated routes.

Kaplan and Prato [10] define the sets used by the researcher (*universalSet* $\supset$ *masterSet* $\supset$ *considerationSet*) and the sets used by the traveler (*universalSet* $\supset$ *awarenessSet* $\supset$ *viableSet*). A *masterSet* is generated using a simulation technique by finding the shortest path after sampling link travel times form a gamma distribution. The *considerationSet* then is derived from the *masterSet* using a

conjunctive heuristic semi-compensatory model: the probability to find *considerationSet* C_n is given by the probability that respondent n uses a specific set of thresholds for the independent variables in the utility maximizing discrete choice process. If an independent variable is out of range w.r.t. a threshold, the corresponding route is not considered. The thresholds are unknown in advance and they are estimated using a maximum likelihood method. *Path* size correction logit (`PSCL`) is used for route selection from the *considerationSet*. The likelihood of the conjunctive model is expressed and maximized using simulation. Hence the method jointly estimates the thresholds for consideration set generation and the beta coefficient for the `PSCL`.

Schüssler et al. [11] cover route choice set generation and validate the results using GPS recorded traces. The `BFS-LE` (Breadth-first-search Link-elimination) technique is presented. `BFS` refers to the fact that a tree of networks is considered and in each network a shortest path is determined using the A* algorithm. The tree is constructed by consecutively eliminating each element from the shortest path such that each generated network differs in exactly one edge from the original one (the parent network). This is done recursively. The `SCSG` (Stochastic Choice Set Generation) is implemented and evaluated. The methods are compared w.r.t computational efficiency, coverage (reproduction of observed recorded OD path), path-size (Ben-Akiva, Bierlaire) value distribution to estimate heterogeneity and hierarchical sequence (road-type sequences). Both `SCSG` and `BFS-LE` are reported to produce similar results but `BFS-LE` is reported to be faster (about 300 times).

Halldórsdóttir et al. [12] focuses on bikers routes. The author evaluates the `BFS-LE` (Breadth-First-Search, link elimination technique), `DSGF` (Double Stochasting Generation Function (Fiorenzo-Catalano) random link attribute selection and random preference threshold) and `B&B` (Branch-and-Bound using *detourFactor*, *timeExcessFactor*, a *directional* constraint and a *similarity* constraint in the evaluation). The `B&B` technique which is reported to perform well on reduced complexity car networks, shows the lowest performance in this research. This could be an indication that `B&B` is quite sensitive to the constraints

used.

2.2. Recursive link based methods

Recursive logit models construct a route from the origin L_O to destination L_D by defining and estimating discrete choice models to select the next link from a node L_i after having constructed the partial route $L_O \rightarrow L_i$. The expected maximal utility generated by the selection of each of the outgoing links needs to be computed. The final route is determined by the dynamic programming technique.

Fosgerau et al. [13] use a discrete choice model to decide at each node about the next link to follow. Dynamic programming is used to find a path. At each node the utility for the partial path from the origin is known by accumulation. The expected utility for the remaining needs to be determined for inclusion in the Bellman value function. It is determined by random utility maximization. The authors prove that this process is equivalent to a path based multinomial logit (MNL) with an infinite number of alternatives. The proposed procedure does not require prior generation of a choice set.

Mai et al. [14] extend the research by Fosgerau et al. [13] by allowing the random terms in the utility functions to be correlated. This is done by multiplying the i.i.d. extreme value type I random variable with a link specific scale factor.

Recursive link based methods require *link additive* attributes to be used in the utility function. Hence, route *complexity* cannot be used. However, collections of predicted routes can be validated by comparing the distribution of the complexity to the one found for recorded traces.

2.3. Route prediction based on traveler mental models

Kazagli and Bierlaire [15] and Kazagli et al. [16] use Mental Representation Items (MRIs) : the authors aim to model the simplifications made by the travelers when representing a route. The set of MRI includes *geographical spans* (zones that may cover a subnetwork) but also *representative geocoded points*

identified by a toponym. The authors consider both discrete choice models to select a particular MRI and models to select sequences of MRI in order to construct a route. In Kazagli et al. [16] the authors suggest to use surveys or GPS traces as sources to identify MRIs and state: *"In the same way that the classical path-based models require some map-matching procedures, we need in the case of such a data source to relate the MRI alternatives with the reported locations or the GPS data."* This is where the *importance* values determined from *route decomposition* proposed in this paper comes into play because automatic detection of MRI is required to make the method feasible in practice.

2.4. Discussion

Recording of GPS traces for travelers and the availability of open map data allow for revealed trips to be map-matched. As a consequence, revealed trips can be expressed as link sequences in transportation networks (or as paths in a graph) for which structural properties can be analyzed.

The results of route choice algorithms used in traffic simulators must expose properties that are statistically similar to the properties of recorded evidence. Classic methods ensure this by considering the numbers of left turns, traffic lights etc in the choice model. However, structural path properties that can be derived from graph theoretic concepts have not been considered until recently; they can be used to enhance the quality of route choice sets. We use new graph-theoretic techniques, here, which have not been used in transportation engineering papers, except for in Knapen [17] and Hartman Ben-Arroyo et al. [18], which contribute to the theory of route choice modelling. Furthermore, analysis of these properties allows for automatic extraction of usage patterns that can be used to enhance new methods like the MRI based method (Kazagli et al. [16]).

2.5. Contributions

This paper extends results about route decomposition that have been reported before. Knapen et al. [1] discuss an algorithm to split paths in a graph

into a minimum number of min-cost paths, or *basic path components (BPC)* (see definition 3.2 in section 3.1). The authors also determine the path complexity and show the distribution of the resulting values observed in several sets of observed GPS traces. Sets of routes predicted by any of the route choice techniques mentioned above should take into consideration the path complexity distribution in order to reflect reality.

Hartman Ben-Arroyo et al. [18] show how to enumerate all possible minimum path decompositions. Knapen [17] introduces the concept of *vertex importance* as the relative occurrence frequency of a node as an intermediate destination in a set of minimum decompositions.

This paper contributes to the research by providing (i) an algorithm to find all minimal shortcuts to a given path in a graph, (ii) an efficient minimum *sequential clique cover enumerator* (which will be defined in section 3.1) for proper interval graphs (required for minimum decompositions enumeration) (iii) a method that generates all possible minimum decompositions of a given path and (iv) a method to determine, for each vertex in a given path P, the relative frequency of use as an intermediate destination in the set of minimum decompositions of P (vertex importance).

The proposed techniques are applied to a set of predicted bikers routes for the city of Amsterdam and results are reported.

3. Method and Algorithms

3.1. Definitions and Basics

We begin with some basic standard definitions from graph-theory. We use definitions and notations for *path,walk, clique, independent set* as in Bondy and Murty [19], see also Hartman Ben-Arroyo et al. [18]. We will then continue to new definitions related to the applications in transportation networks.

Let $G = (V, E)$ be a directed graph with vertex set V and edge set E. The vertices correspond to *nodes* in a road network, and the edges correspond to *links* in the network. Each edge e has a non-negative *cost* $c(e)$ which is the

effort (e.g. time or money) required to traverse the link in the network. We denote by $N(v)$ the set of neighbours of vertex v, i.e. the set of vertices adjacent to v. We denote by $deg(v)$ the degree of a vertex v, i.e. $|N(v)|$, the number of neighbours it has. A *path* is a walk where all its vertices are distinct. For a path $P = (v_0, v_1, \ldots, v_l)$, any subsequence of vertices $v_i, v_{i+1}, \ldots, v_j$, where $0 \leq i \leq j \leq l$ is a *subpath* of P, and is denoted by $P(v_i, v_j)$. The *length* of a path, is the number of edges in it (i.e. l), the *size* of a path, denoted by $|P|$, is the number of vertices in it (i.e. l+1), and the *cost* of a path, denoted by $c(P)$ is the sum of the costs of its edges. A path $P(v_0, v_l)$ is a *least cost path* between v_0 and v_l, if there exists no other path connecting v_0 and v_l of lower cost.

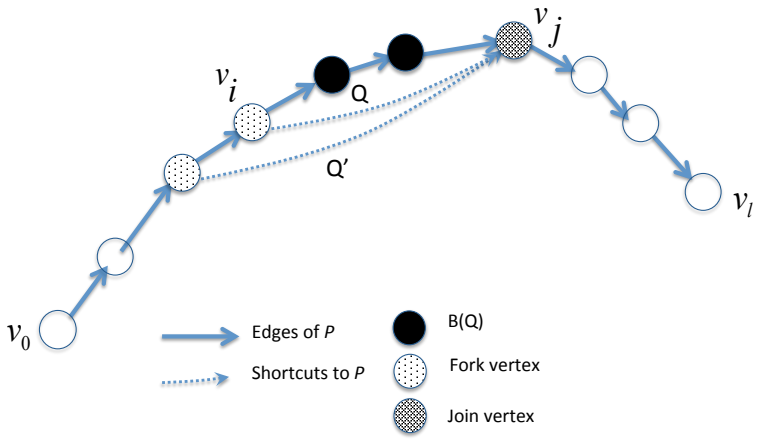
We remark that if $c(e) = 1$ for all $e \in E$ then the cost of a path coincides with its size. We assume that the vertex traversal cost is zero. A single edge (u, v), being a path connecting between u and v, may be least cost, or not. If it is not a least cost path connecting between u and v, then it is called a *non-least-cost edge.*

It is easy to see that if P is a least cost path, then any subpath of P is also a least cost path.

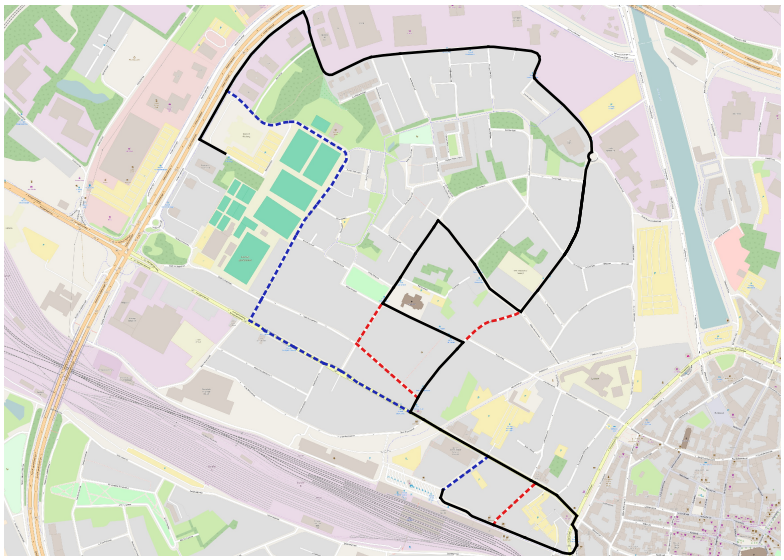
The converse of this statement is false since it is possible that all the subpaths of $P(v_0, \ldots, v_l)$ (except P itself) are least cost paths, but P is not a least cost path connecting v_0 and v_l and there is another least cost path Q connecting v_0 and v_l. This fact motivated the following definition, as in Knapen et al. [1].

Definition 3.1 (P- shortcut, minimal shortcut, fork and join vertices, bypassed vertex set)**.** *Let $P = (v_0, v_1, \ldots, v_l)$ be a given path. A $P(v_i, v_j)$-*shortcut *(or for brevity, P -* shortcut*, or* shortcut*), is a path $Q(v_i, v_j)$, internally- disjoint from P, where $v_i, v_j \in V(P)$, such that $c(Q(v_i, v_j)) < c(P(v_i, v_j))$. The vertices v_i and v_j are called* fork *and* join *of the shortcut, respectively, and the internal vertices of P between the fork and the join (i.e. $v_{i+1}, \ldots, v_{j-1}$) are called Q-*bypassed vertex set*, or* bypassed vertex set *and denoted by $B(Q)$. A shortcut Q is* minimal *if $B(Q)$ does not properly contain $B(Q')$ where Q' is another shortcut to P (See Figure 1).*

We emphasize that $B(Q)$ contains consecutive vertices on P. Therefore, it can be marked by the fork and join of a shortcut Q, which are the vertices preceding, and following the set $B(Q)$, respectively.



(a) A path P with a minimal $P(v_i, v_j)$-shortcut Q. Q' is a non-minimal shortcut. Vertices v_i and v_j are fork and join vertices of Q, respectively, and the dark vertices are the bypassed vertex set $B(Q)$.



(b) A path in the road network. The dashed lines represent shortcuts (red:minimal, blue:non-minimal).

Figure 1: (a) Shortcut concepts and (b) Shortcuts in a real path.

Clearly, a least cost path cannot have any shortcuts.

Definition 3.2 (Basic Path Component (BPC), path splitting, splitVertex)**.**

Given a path P, a subpath of P is called a Basic Path Component*, or for short, a BPC, if it is either a least cost path connecting its endpoints, or P is a single non-least-cost edge. A* path splitting *of P is a partition of P into subpaths each of which is a basic path component. A* splitVertex *is a vertex separating two consecutive BPC in a path splitting.*

We remark that there may be many ways to split a path, for example, the trivial partition into single edges $(v_0, v_1), (v_1, v_2), \ldots, (v_{l-1}, v_l)$ is an example of such a partition. We are interested in finding a path splitting with a *minimum number* of basic path components. Such a path splitting is called *minimum path splitting*. Each non-shortest-edge is a part in each minimum path splitting since it constitutes a BPC, the vertices of a non-shortest-edge are splitVertices. If we remove the set of non-shortest-edges in a path (each of which is a BPC), we are left with a set of disjoint paths, each of which contains no non-shortest-edges.

In Knapen et al. [1] we addressed the problem of finding efficiently a minimum path splitting of a given path. Since a minimum path splitting will contain a minimum number of splitVertices, an equivalent formulation of the problem above is to find a minimum number of splitVertices in the path, such that any subpath connecting consecutive splitVertices will be least cost. The following lemma was proved in Hartman Ben-Arroyo et al. [18] :

Lemma 3.3. *If P is not least cost (and not a non-least-cost edge) with a shortcut Q, then any path splitting of P will contain at least one vertex in $B(Q)$, the Q-bypassed vertex set, as a splitVertex.*

From lemma 3.3 it immediately follows that a minimum path splitting of P is obtained by a minimum set of splitVertices which intersects every bypassed vertex set, $B(Q)$, for all minimal shortcuts Q to P.

3.2. Technique Overview

An overview of the technique is summarized in Figure 2. (A) shows a path P in the transportation graph representing the road network along with some minimal shortcuts to the path. Each minimal shortcut bypasses some vertices

on the given path (for example v_x and v_y in the graph G^T). In the top part of (B), the vertices in P are represented by integer points on the horizontal axis (which are not marked here). Each sequential set of vertices bypassed by a minimal shortcut constitutes an interval represented by a line segment (labeled $a, b, c, \ldots$). For simplicity, we marked only the endpoints of the intervals. The corresponding interval intersection graph is shown in the bottom part. (C) shows some of the minimum sequential clique covers for the interval intersection graph.

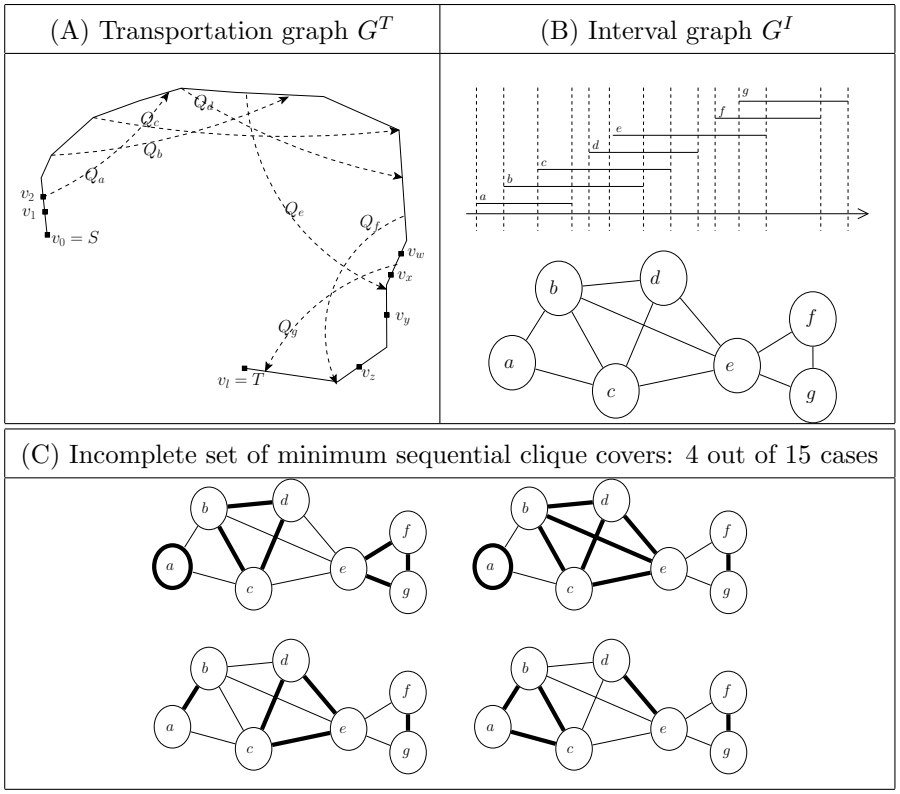


Figure 2: Overview of graphs used to enumerate minimum path decompositions.

The algorithmic steps used to extract path properties and vertex importance for a path P are as follows:

1. Finding all non-least-cost edges, and all maximal subpaths of P which do not contain any non-least-cost edges using the algorithm described in

Knapen et al. [1].

2. Finding all minimal shortcuts in each of the subpaths found above. Each minimal shortcut defines a sequence of bypassed path vertices. Each bypassed vertex set is mapped to an interval. (See section 3.3).

3. Defining the intersection graph of the intervals defined above. This leads to a proper interval graph (indifference graph) since none of the bypassed vertex sets is included in another. (See section 3.4).

4. Defining the proper interval graph G^I, and sequential clique covers, (or s-clique covers, for short). (See section 3.4).

5. Enumeration of minimum s-clique covers for each connected component, (see section 3.5).

6. Enumerating all path decompositions (see sections 3.6 and 3.7).

7. Computation of vertex importance values. (See section 3.8)

The non-trivial steps are detailed in the following subsections.

3.3. Finding all minimal shortcuts and non-least-cost edges

In the first stage we find all minimal shortcuts to P, as was described in Hartman Ben-Arroyo et al. [18]. We repeat the algorithm for the sake of completeness. We do not need the shortcut paths to P, but rather their endpoints, the fork and join vertices of each shortcut. The output is a list of pairs $< v_f, v_j >$ corresponding to the fork and join vertices of all minimal shortcuts, or of non-least-cost edges.

Assume a traveler moves from point v_0 to point v_l along a path $P = (v_0, v_1, \ldots, v_l)$. Dijkstra's [20] shortest path algorithm is used to find the first vertex on P, (if it exists), say v_j, for which $P(v_0, v_j)$ is not the shortest path connecting v_0 and v_j. If such a vertex does not exist, the given path is a shortest path. Otherwise, we mark v_j as a join vertex, and continue by finding the *last* vertex in the subpath $P(v_0, v_j)$, say v_f for which $P(v_f, v_j)$ is not a least-cost

path. We output the shortcut $< v_f, v_j >$ and continue with the subpath of P beginning with v_{f+1} . The pseudo code is given in algorithm 3.1.

Algorithm 3.1 Minimum shortcut and non-least-cost edge finder

Require: graph G, path P in G

```
1: function FINDFIRSTJOIN(P, from) ▷ Search first join following vertex from
2:     join ← succ(P, from)
3:     while (join ≠ nil) ∧ (shortPath(P, from, join)) do
4:         join ← succ(P, join)
5:     end while
6:     return join
7: end function
```

```
8: function FINDLASTFORK(P, from)      ▷ Search first fork preceding vertex
   from
9:     fork ← pred(P, from)
10:    while (fork ≠ nil) ∧ (shortPath(P, fork, from)) do
11:        fork ← pred(P, fork)
12:    end while
13:    return fork
14: end function
```

Line 27 is used to minimize the work. At that point it is known that no minimal shortcut forks in a vertex $v \in [dsucc(start), v_j]$.

3.4. Defining the intervals, the proper interval graph G^I, and sequential clique covers

The proper interval graph is defined as in Hartman Ben-Arroyo et al. [18]. We repeat the definition for the sake of completeness. We assume for now that P is a path containing no non-least-cost edges. We may assume that since any

15: $start \leftarrow v_0$

16: **while** $start \neq$ **nil do**

17: $\quad v_j \leftarrow \mathit{findFirstJoin}(P, start)$

18: $\quad$ **if** $v_j =$ **nil then**

19: $\quad\quad start \leftarrow$ **nil** ▷ Path exhausted

20: $\quad$ **else if** $v_j = \mathit{succ}(P, start)$ **then** ▷ Non-least-cost edge detected

21: $\quad\quad \mathit{output}('\text{nonLeastCostEdge}', \langle start, v_j \rangle)$

22: $\quad\quad start \leftarrow v_j$ ▷ Skip over non-least-cost edge

23: $\quad$ **else**

24: $\quad\quad v_f \leftarrow \mathit{findLastFork}(P, v_j)$ ▷ Regular shortest

25: $\quad\quad$ **if** $v_f = \mathit{pred}(P, v_j)$ **then** ▷ Non-least-cost edge detected

26: $\quad\quad\quad \mathit{output}('\text{nonLeastCostEdge}', \langle v_f, v_j \rangle)$

27: $\quad\quad\quad start \leftarrow v_j$ ▷ Skip over non-least-cost edge

28: $\quad\quad$ **else**

29: $\quad\quad\quad \mathit{output}('\text{minShortCut}', \langle v_f, v_j \rangle)$

30: $\quad\quad\quad start \leftarrow \mathit{succ}(P, v_f)$ ▷ $start \neq v_j$

31: $\quad\quad$ **end if**

32: $\quad$ **end if**

33: **end while**

path, by removing the non-least-cost edges, breaks down to subpaths which contain no non-least-cost edges. Once we know all minimal shortcuts to P, we use the corresponding bypassed vertex sets to define a set of intervals and a corresponding interval graph. Since the vertices of P are labeled $v_0, v_1, v_2, \ldots, v_l$, every shortcut Q with fork and join vertices v_f, v_j, respectively has a consecutive set of bypassed vertices $B(Q) = \{v_{f+1}, v_{f+2}, \ldots, v_{j-1}\}$. We construct an interval on the real line corresponding to each bypassed vertex set in the following way: $I_Q = [f+1, j-1]$ (see Figure 2 (B)). Note that the integral points on the interval $[f+1, j-1]$ (i.e. the points $f+1, f+2, \ldots, j-1$) correspond to the vertices $v_{f+1}, v_{f+2}, \ldots, v_{j-1}$ on P. Since the shortcuts found in Stage 1 are minimal shortcuts, no two intervals contain each other. The intersection graph of such a set of intervals, where no two intervals contain each other, is called a *proper interval graph*, or equivalently, *unit interval graph*, or *indifference graph* (see Golumbic [21]). We denote it by $G^I = (V^I, E^I)$, where each $v \in V^I$ corresponds to $B(Q)$ of some shortcut Q, and two vertices are adjacent if and only if the corresponding intervals intersect.

We note that if we order all the intervals representing V^I by their left hand endpoint, in increasing order, then, being a proper interval graph, their right hand endpoints will also be in increasing order (otherwise one interval will contain another). A *clique* in a graph is a subset of vertices all of which are adjacent to each other, i.e. a complete subgraph. A *clique cover* is a set of cliques which cover all the vertices in the graph. A *minimum clique cover* is a clique cover which contains a minimum number of cliques. For any point on the real line, the set of intervals which contain that point, mutually intersect each other, and therefore correspond to a *clique* in G^I. A minimum set of points which meet all the intervals will correspond to a minimum clique cover of G^I. In order to enumerate all minimum decompositions of P, we need to enumerate all the minimum sets of integer points which cover all the intervals. Note that given the ordered set of intervals representing V^I, any integer point - corresponding to a vertex of P - is contained in a *consecutive* set of intervals which mutually intersect each other. We call such a clique a *sequential clique*, or for short,

s-clique. The s-cliques are naturally ordered so that each interval belongs to consecutive cliques , and each s-clique consists of consecutive intervals. For example, the clique (abc) in Figure 2 is an s-clique, and the cliques (ac) or (bce) are not s-cliques.

3.5. Finding all minimum s-clique covers for proper interval graphs

Assume the intervals are given and they are labelled by $1, 2, ..., n$, where the order is by the left hand endpoints of the intervals. The corresponding vertices of the graph are also labeled $1, 2, ..., n$. (We remind the reader that each interval corresponds to the bypassed vertex set of a shortcut.) We assume the all shortcuts are minimal under inclusion, in other words no interval contains another. For any *consecutive* subset of intervals $S \subseteq V$ we define a function $f(S)$ which computes the *clique covering number* of the graph $G[S]$, i.e. the minimum number of cliques required to cover S. We compute this function recursively. We use the function f in another recursive function $GenMCC(V)$ which generates all minimum s-clique covers with respect to the intervals $1, 2, ..., n$.

Algorithm 3.2 Compute the clique covering number of proper interval graph $G[V]$.

```
1: function f(V)        ▷ The vertices are ordered by their left hand endpoints
2:     if V = ∅ then
3:         return 0
4:     else
5:         Let v be the first vertex in V
6:         return 1 + f(V \ {v} \ N(v))
7:     end if
8: end function
```

Claim 3.4. *The function in 3.2 finds the clique covering number of the graph* $G[V]$.

Proof. We assume the vertices in V are ordered by the left hand endpoints of the intervals representing them. If the intervals are not given, we can find

a *perfect elimination ordering* of G, which will correspond to the left hand endpoint ordering of the intervals. Algorithm 3.2 finds a maximal clique that contains the first vertex, (consisting of v and all its neighbours), removes it, and continues recursively for the rest of the graph. Assume the function returns the number c. Then in each call of the function the set $\{v\} \cup N(v)$ is a clique and all c cliques form a clique partition of $G[V]$. Note also that the set of all first vertices in all the calls form an independent set of the same size c. This proves that the clique partition is minimum. □

The following recursive function generates all minimum s-clique covers, and replaces Stage 3 in the algorithm described in Hartman Ben-Arroyo et al. [18] .

Algorithm 3.3 Generate all minimum s-clique covers of proper interval graph $G[V]$.

```
1: function GenMCC(V)        ▷ The vertices are ordered by their left hand endpoints
2:     Let 1 be the first vertex in V, let d = deg(1) in G[V]
3:     if d + 1 = |V| then                ▷ Graph consists of a unique clique
4:         return {V}
5:     end if
6:     for all x ∈ {2, ..., d + 2} do
7:         if f(V \ {1, ..., x − 1}) = f(V) − 1 then
8:             for all x − 1 ≤ j ≤ d + 1 do
9:                 return {{1, 2, ..., j}} ∪ GenMCC(V \ {1, ..., x − 1})
10:            end for
11:        end if
12:    end for
13: end function
```

Theorem 3.5. *Algorithm 3.3 generates all minimum s-clique covers of the graph $G[V]$.*

Proof. We will first show that every minimum s-clique cover is generated by the algorithm. We will then show that every minimum s-clique cover is generated exactly once.

We consider the first (leftmost) interval, labeled 1. Any clique cover must cover it. If interval 1 and its neighbours consists of the whole graph then the function returns one clique cover which consists of that clique V, as in line 4. Otherwise, $d+2$ denotes the first interval which is not in $N(1)$. Every minimum s-clique cover of size q contains a clique $C_1 = \{1, 2, ..., j\}$ for some $j \leq d+1$ and $q-1$ cliques which cover $x, x+1, \ldots, |V|$ for some $2 \leq x \leq d+2$. In line 7 we verify that the clique covering number of $G[x, x+1, \ldots, |V|]$ is indeed $q-1$, and in line 8 we guarantee that C_1 covers at least the vertices $1, 2, \ldots, x-1$, so that every vertex is covered by the clique covering.

To show that every minimum s-clique covering is generated exactly once, note that for any ordered pair (j, x) such that $x-1 \leq j \leq d+1$ and $f(V \setminus \{1, ..., x-1\}) = f(V) - 1$ defines a unique minimum s-clique cover.

We note that this algorithm is highly efficient, in fact it is a polynomial delay algorithm. To see this, note that whenever the condition in line 7 is met, a family of solutions is generated for each $x-1 \leq j \leq d+1$, and in order to satisfy this condition only a linear time search is employed, as seen in line 6. □

3.6. Path decomposition enumerator

In this section we enumerate the number of ways of breaking a path into basic path components, by considering the bypassed vertex sets. We assume that G^I is connected, otherwise we apply the algorithm for each connected component.

3.6.1. Definitions: Clique core and Core collection

In order to define the split vertex selection procedure, the *core of a clique* concept is introduced.

Let $C_{i,j}$ denote the clique containing intervals $I_i, I_{i+1}, \ldots, I_j$.

Definition 3.6 (Clique core)**.** *The* core of a clique *or* clique core*, denoted by $\mathcal{K}(C_{i,j})$, is the set of integer points of the real line contained in those intervals*

only, and in no other intervals. In other words, the core of a clique corresponds to the set of bypassed vertices associated with the clique $C_{i,j}$ and to no other bypassed vertex sets.

The example shown in Figure 3 applies to the path shown in Figure 2. $\mathcal{K}(ef)$ contains the integer w corresponding to the bypassed vertex v_w, $\mathcal{K}(efg)$ contains x which corresponds to the bypassed vertex v_x and $\mathcal{K}(fg)$ contains the points which correspond to v_y and v_z.

Definition 3.7 (Core collection)**.** *We denote an s-clique cover by $\mathcal{C}$ and the collection of clique cores in $\mathcal{C}$ by $\mathcal{K}(\mathcal{C}) = \{\mathcal{K}(C_{i,j})\}$ with $C_{i,j} \in \mathcal{C}$. We call it* core collection of $\mathcal{C}$.

3.6.2. Enumeration procedure for subpaths containing no non-least-cost edges.

For a given path P in the transportation network, enumeration of all minimum path decompositions is done by (i) considering all minimum s-clique coverings $\mathcal{C}$ of the corresponding interval graph G^I, (ii) determining the corresponding core collection $\mathcal{K}(\mathcal{C})$ for each $\mathcal{C}$ and (iii) determining a minimum decomposition of P by taking precisely one vertex from each core in $\mathcal{K}(\mathcal{C})$ as a splitVertex (see Lemma 3.3). The *splitvVertices* in each set selected by this procedure partition P into BPC because at least one bypassed vertex is chosen for each minimal shortcut. Furthermore, the path decompositions are minimum because minimum clique coverings are used. It is easy to verify that the path decompositions are unique as well, i.e. different minimum clique coverings have different core collections. For a given clique cover, since the choice of every core vertex is independent of the choice of other core vertices, hence the number of possible path splittings for a given path is,

$$\sum_{\mathcal{C}\in\mathbb{C}} \left(\prod_{C_i\in\mathcal{C}} |\mathcal{K}(C_i)| \right) \tag{1}$$

where $\mathbb{C}$ denotes the set of all minimum s-clique covers in G^I, $\mathcal{C}$ denotes a specific clique cover, and each $C_i \in \mathcal{C}$ represents a sequential clique, i.e. s-clique.

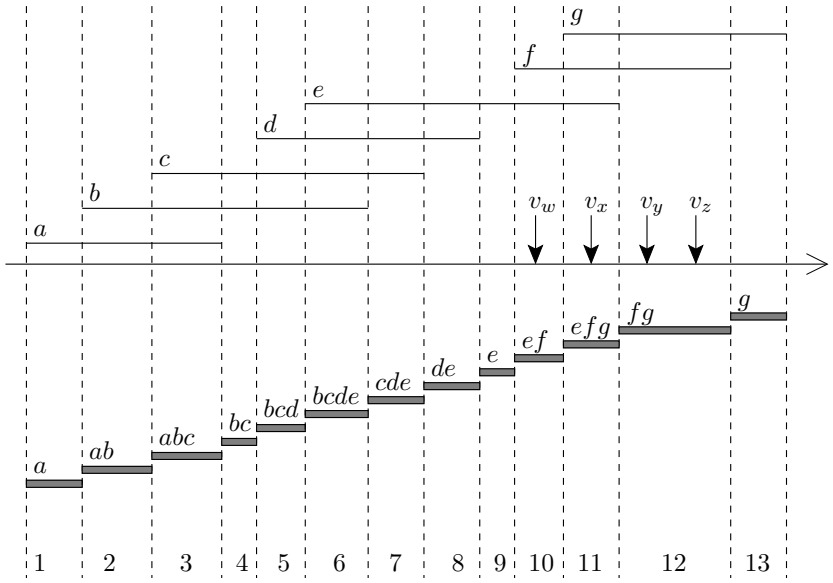


Figure 3: Some of the *core*s for the path shown in Figure 2

3.7. Path decomposition enumeration

Sections 3.3 to 3.6 apply to paths which do not contain any non-least-cost edges. In the final step, we prepare to generate all minimum decompositions for the complete path. First we remind that each vertex belonging to a non-least-cost edge is a split vertex in a minimum decomposition unless it is the first or the last vertex in the path.

Second, we consider bypassed vertex sets. All core collections for a given path have the same size since they are associated with minimum s-clique coverings. However, in some specific cases, the core of a clique in a minimum s-clique cover may be empty; (this occurs when a clique consists of a single vertex which corresponds to an interval of length one. In this case the interval contains no integer points except for its endpoints. An example is shown in Figure 4). In this case we ignore the s-clique cover.

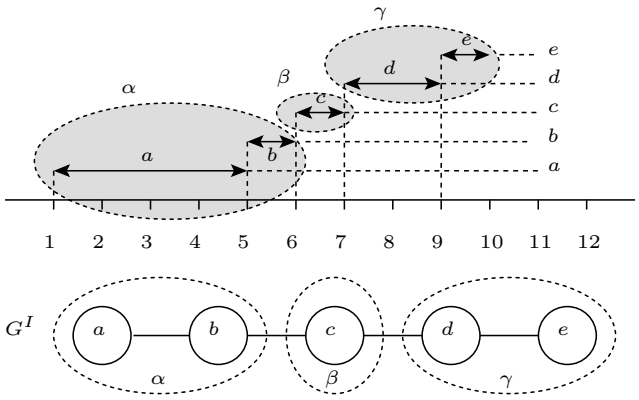


Figure 4: Top: bypassed vertex sets (corresponding to intervals a, b, c, d, e). Bottom: the corresponding intersection graph G^I and one of its minimum clique coverings. Clique $\beta = \{c\}$ has an empty core: $\mathcal{K}(\beta) = \emptyset$. The clique covering $\{\alpha, \beta, \gamma\}$ does not generate any minimum decomposition.

Consider a path P in the transportation graph. Let E' denote the set of non-least-cost edges in P, and denote by W the set of vertices of E', not including the endpoints of the path P. If we remove E' from P we get a collection of vertex disjoint subpaths $P_1, P_2, \ldots, P_l$. Each such subpath P_k, if it contains shortcuts, produces an interval graph G^I and a corresponding family of minimum s-clique covers $\mathbb{C}_k$. In order to find all minimum decompositions of P, we consider all combinations of s-clique covers of every subpath P_k, in addition to the set W which needs to be covered by singleton vertices. Every such clique cover is called a *Minimum Decomposition Generator (MDG)*. Recall that every clique $C_i \in \mathcal{C} \in \mathbb{C}$ corresponds to a core, and any vertex from the core is a splitVertex in some minimum path decomposition, hence every clique cover may produce a large number of minimum path decompositions. See Figures 6, 7 and 8 for MDG's which are denoted by orange intervals.

3.8. Vertex importance

We assume that the higher the frequency of use of a vertex as a *splitVertex* in a decomposition, the higher the probability that it carries a meaning (as an intermediate destination) relevant to the traveler. This assumption is similar to

what is done in the *trajectory annotation* process (i.e. the process that tries to assign a meaning to each *stop* detected in a GPS trace). In that process, the visit frequency and the total time spent at a given location are quantities used while trying to discover the intention of a stop.

In a similar way, in the case of route splitting, we try to find the probability for a vertex to be the *splitVertex* that the user had in mind.

Definition 3.8. *The* path based importance $i_p(v, P)$ *of a vertex* v *in a path* P *is the relative occurrence frequency of* v *as a* splitVertex *in the set of all minimum decompositions of* P*.*

Let $\mathcal{D}(P)$ denote the set of all minimum decompositions of P. Let $\mathcal{D}_v(P)$ denote the set of all minimum decompositions of path P containing *splitVertex* v. Then

$$\mathcal{D}_v(P) = \{d \in \mathcal{D}(P) \mid v \text{ is a splitVertex in } d\} \tag{2}$$

$$i_p(v, P) = \frac{|D_v(P)|}{|D(P)|} \tag{3}$$

For example, the vertices incident to a non-least-cost edge have path importance 1 since every minimum decomposition into BPC's includes them as a splitVertex. If v is not a vertex of P then $i_p(v, P) = 0$. It may be relevant to consider indicator quantities relative to particular sets of paths e.g., all paths emerging from a particular region. For a family of paths $\mathcal{P}$, we denote by $V[\mathcal{P}]$ the set of vertices covered by $\mathcal{P}$. We define the *normalized path family importance* of a vertex v in a family of paths $\mathcal{P}$, denoted by $i_f(v, \mathcal{P})$, by summing the path based importance values for each vertex covered by paths in $\mathcal{P}$ and dividing by the maximum of these sums taken over all $v \in V[\mathcal{P}]$.

$$i_f(v, \mathcal{P}) = \sum_{P \in \mathcal{P}} i_p(v, P)/M \tag{4}$$

where

$$M = \max_{v \in V[\mathcal{P}]} \Big\{ \sum_{P \in \mathcal{P}} i_p(v, P) \Big\} \tag{5}$$

Note that $i_f(v, \mathcal{P})$ does not directly depend on the number of paths in the family $\mathcal{P}$ and is normalized to the interval $[0, 1]$. This is done because the importance indicator for a vertex v should not be affected by the mere presence of paths in $\mathcal{P}$ not containing vertex v.

Interesting importance values can be calculated by restricting the family $\mathcal{P}$ of paths considered (e.g. the paths for a given individual, the paths having a given destination, the paths that connect two given areas, the paths for which the trip execution overlaps a given time window, etc).

3.9. Evaluation of $i_p(v, P)$, the path based importance of a vertex

Enumeration of all minimum path decompositions may be a costly operation in practice. We observe that the contribution of an MDG to the path importance is the same for all vertices in a core. The computation of path importance does not require the explicit enumeration of all possible decompositions of a path but takes advantage of the structural properties of MDGs.

Consider an MDG and some s-clique in G^I. This can be computed as follows: let $\mathcal{M}(P)$ be the set of all MDG for path P and let μ_k denote the k-th MDG in $\mathcal{M}(P)$. Let $\mu_k[j]$ denote the j-th core in the k-th MDG and $\mid \mu_k[j] \mid$ denotes its cardinality. Remember that the number of cores in each MDG for path P equals the *complexity* of P, which equals the clique covering number of the interval graph G^I corresponding to P. Similarly to Equation (1), the number of minimum decompositions for P is given by

$$N = \sum_{k \in \mid \mathcal{M}(P) \mid} \Big(\prod_{j=1,\ldots,c(P)} \mid \mu_k[j] \mid \Big) \tag{6}$$

where $c(P)$ denotes the complexity of P. The contribution to the path based importance for all vertices v in the m-th core of μ_k is the same and is given by

$$\frac{\prod_{j=1,..,m-1,m+1,..,c(P)} \mid \mu_k[j] \mid}{N} \tag{7}$$

We define a *location* function $L(k, v)$ that gives the index in μ_k of the interval

which contains v, if any, and $c(P)+1$ otherwise:

$$L(k,v) = \begin{cases} i & if \quad \exists i \in [1, c(P)] \mid v \in \mu_k[i] \\ c(P)+1 & otherwise \end{cases} \tag{8}$$

A two-dimensional $|\mathcal{M}(P)| \times (c(P)+1)$ contribution matrix $\mathbf{K}$ is defined as:

$$\mathbf{K}(k,m) = \begin{cases} \prod_{j=1,..,m-1,m+1,..,c(P)} \mid \mu_k[j] \mid & if \quad m \leq c(P) \\ 0 & if \quad m = c(P)+1 \end{cases} \tag{9}$$

Then, the path based importance can be written as

$$i_p(v,P) = \frac{\sum_{k \in |\mathcal{M}(P)|} \mathbf{K}[k, L(k,v)]}{N} \tag{10}$$

which allows for efficient computation of path importance of a vertex without explicitly enumerating all decompositions.

4. Experiment

The technique described in section 3 was implemented by the authors of this paper. A fully operational software tool for path decomposition and importance evaluation was created. Its application is described in the following subsections.

4.1. Data

1. The aim of this paper is to show the newly developed technique and to evaluate the computing performance. The intention was not yet to analyze recorded GPS traces and draw conclusions about revealed behavior. A set of origin-destination pairs for bikers routes in Amsterdam was available but no routes extracted from GPS traces.

2. In order to produce test data, the `POSDAP` software described by Montini and Schüssler [22] was used to generate routes from the available origin-destination pairs in order to demonstrate the technique described above. More specifically, the bike route predictor from `POSDAP` was used to generate synthetic input data for the experiment. No other use of `POSDAP`

software was made in the experiment. Predictions are generated using the *doubly stochastic choice set generator* for bike routes. Both the link characteristics and the cost function parameters are stochastically sampled in successive trials to generate a new route. Then the least cost path for the specified origin-destination pair is determined and added to the choice set if it is a new one. The required size of the choice set is specified to `POSDAP` by a parameter. The choice set generator cannot guarantee that sufficient different paths can be found for any particular origin-destination pair and stops after a pre-specified effort. The original `POSDAP` software by ETHZ was slightly adapted to stop the trials after a predefined number of trials instead of by exceeding a predefined computing time in order to produce the same results for the same input on different machines.

3. Figure 5 shows the predictions for two origin destination pairs. The routes for the respective choice sets are shown in wide lines in different colors. The thin lines in the background show the street network in Amsterdam.

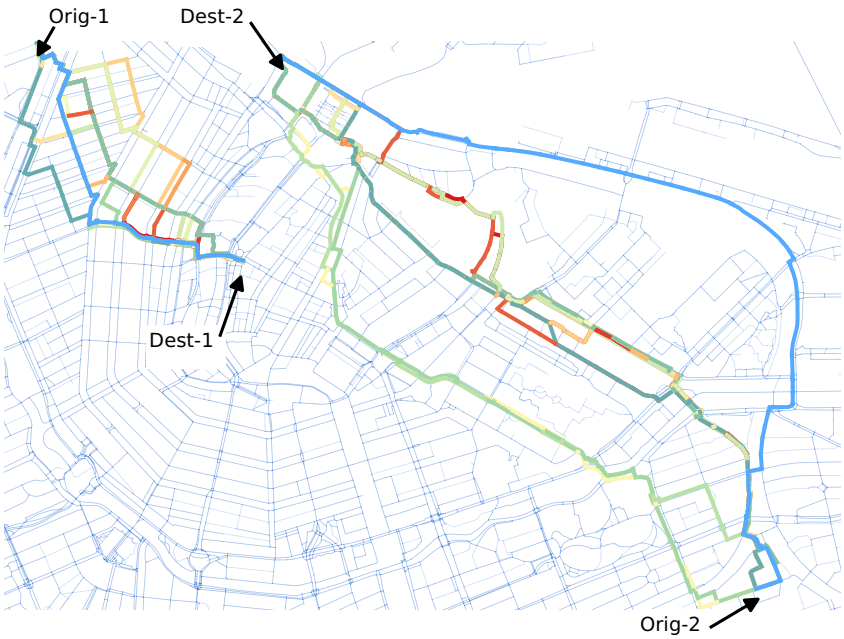


Figure 5: Choice sets for two origin destination pairs in Amsterdam generated by the `POSDAP` doubly stochastic choice set generator for bike routes

4. The code is written in `Java7`. All machines ran identical code. The code is written in `Java7`. Four different computers were used to execute performance evaluation experiments. These apply solely to the *path decomposition* and *importance evaluation* techniques (the input data generation using `POSDAP` was done in advance). The resulting values are shown in table 1. The values comprise all steps in the process for each given path: (i) finding non-least-cost edges, (ii) determination of the size of the minimum path decomposition, (iii) finding all minimal shortcuts to the path, (iv) determination of the interval graphs, (v) determination of all minimum s-clique covers, (vi) creation of minimum decomposition generators and (vii) computing the importance for each split vertex. This takes between 1.0 and 2.2 seconds per given path (depending on the machine used). All machines ran identical code.

Machine Name	Processor	ClockFreq [GHz]	Memory [GB]	OS	nThreads -	nRoutes -	Runtime [sec]	routes/sec [1/sec]
imobcalc3	Intel(R) Xeon(R) CPU X5570	2.93	48	14.04.1-Ubuntu SMP	8	98733	97548	1.012
imobcalc7	Intel(R) Xeon(R) CPU E5-2660	2.00	128	SMP Debian (jessie)	8	99090	60384	1.640
imobcalc8	Intel(R) Xeon(R) CPU E5-2630	2.20	128	Ubuntu SMP	8	98304	45959	2.138
sd-124739	Intel(R) Xeon(R) CPU D-1531	2.20	32	SMP Debian (jessie)	8	99147	88404	1.121

Columns specification

nThreads : Number of `Java` threads used.

nRoutes : Number of routes processed . The routes were nearly evenly distributed over the machines.

Runtime : Difference in system times recorder at start and termination

routes/sec : Number of routes programmed per second

Table 1: Machine specifications and performance indicators

4.2. Results

For three typical paths the *bypassed vertex sets* and *minimum decomposition generators (MDG)* are shown in diagrams (Figures 6, 7 and 8). The integer numbers on the horizontal horizontal axis correspond to the offsets of the vertices in the path (first vertex has offset zero). Components in the diagram are drawn at layers labeled by an integer number on the vertical axis (the values near both axes are to be interpreted as labels, not as quantities).Each block (point) at layer 0 represents a vertex of the given path. Endpoints of non-shortest edges are represented by green blocks, the other vertices by red blocks.

The blue line segments represent bypassed vertex sets. All vertices at layer 0 that are covered by a blue segment belong to the same bypassed vertex set. For clarity the line segments have been drawn at different layers in order to avoid overlap so that it is easy to see for each vertex by which minimal shortcuts it is bypassed.

Each layer with a negative label corresponds to one MDG. An orange line segment represents a core (containing all vertices at layer 0 covered by the line segment). In each case, only the first seven MDG are shown in the diagrams (to avoid clutter). Exactly one vertex is to be chosen from each core at a particular layer. Note that each orange layer has the same number of cores. The number of decompositions generated by the MDG represented by a particular layer equals the product of the number of vertices in the cores identified by line segments in that layer.

A black filled circle at horizontal position i and at the vertical position between layers labeled 0 and -1 represents a vertex in the path; the diameter measured along the horizontal axis equals the normalized path family importance for v_i in the set of all paths (some are very small).

The meaning for the headers in the table below is as follows:

- **Largest Clique *Covering* Number**: size of the largest minimum s-clique cover found among all connected components of the bypassed vertex sets intersection graph (definition: see section 3.5)

- **Clique Number**: the size of the largest clique of G^I. This corresponds to the maximum number of minimal shortcuts a vertex in the transportation graph G^T is bypassed by.

- **#non-least-cost edges**: the number of *non-least-cost edge*s.

- **#MDG**: the number of *minimum decomposition generators* that collectively enumerate all minimum decompositions for the path.

Figure	Largest Clique Covering Number	Clique Number	#non-least-cost edges	#MDG
6	2	4	1	10
7	3	2	0	5 (all shown)
8	2	6	0	63

The Figures 6, 7 and 8 show patterns that do occur frequently. About 60% of the cases are simpler then the ones shown.

Figure 6 shows a case containing a non-least-cost edge (between vertex offsets 11 and 12).

The case in Figure 7 shows a small clique number. It has only 5 minimum decomposition generators all of which are shown.

The pattern in 8 has a moderate number of bypassed vertex intervals but a large clique number. It has 63 MDGs which is among the largest values that have been observed. This leads to small importance values and hence the representing circles are nearly invisible.

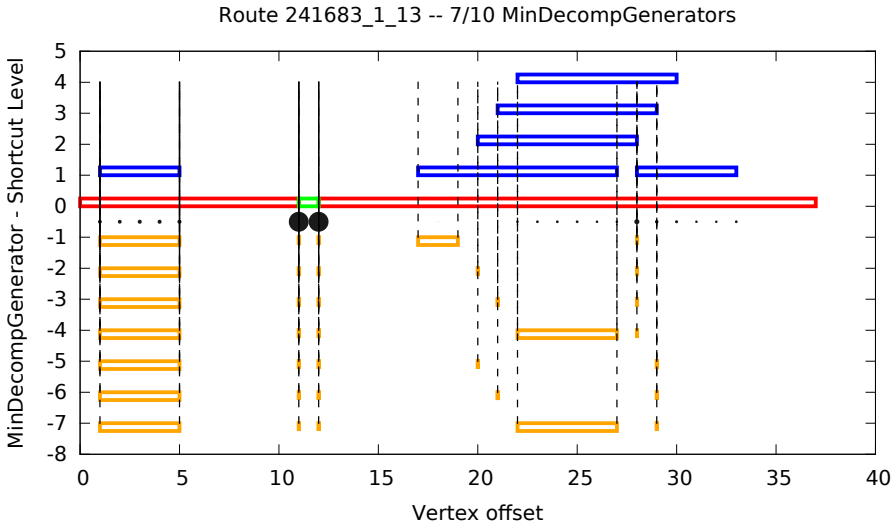


Figure 6: Legend: (A) Each block at layer 0 corresponds to a vertex in the path (green: vertex from non-least-cost edge, red: other vertices). The vertex offset in the path can be read from the values near the horizontal axis. (B) Each (blue, orange) line segment identifies a set of consecutive vertices on the path. That set contains exactly each vertex at layer 0 covered by the line segment. (C) Blue line segments represent bypassed vertex sets. They are drawn non-overlapping and using a minimum number of layers in order to clearly show by which minimal shortcuts each vertex is bypassed. (D) Orange line segments represents *cores*. The cores at a given layer together constitute a single MDG. (E) The radius of the black circles shown between layers 0 and -1 is proportional to the importance of the vertex shown right above the circle. Normalized path family importance is shown. Some of the circles are very small.

For this case the orange layers represent 7 out of the 10 MDG found. The clique number = 4.

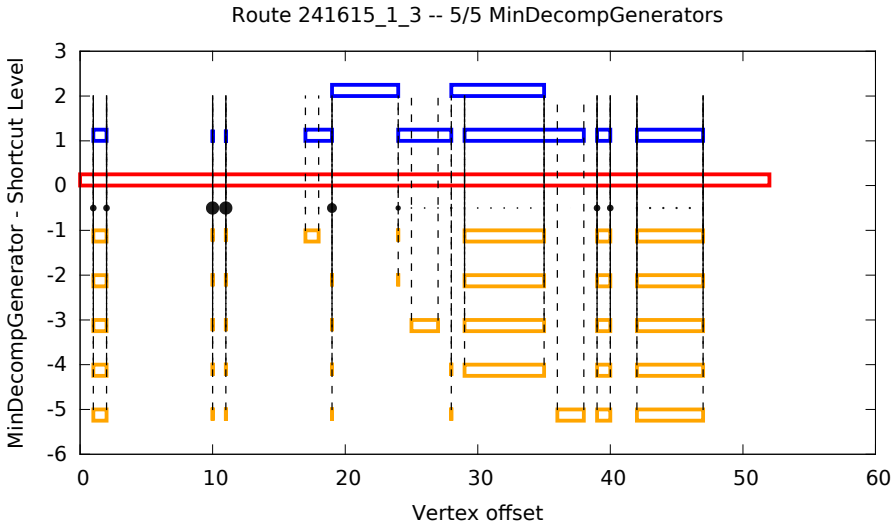


Figure 7: The legend is explained in the caption of Figure 6. Clique number = 2. The orange layers represent the complete set of MDG.

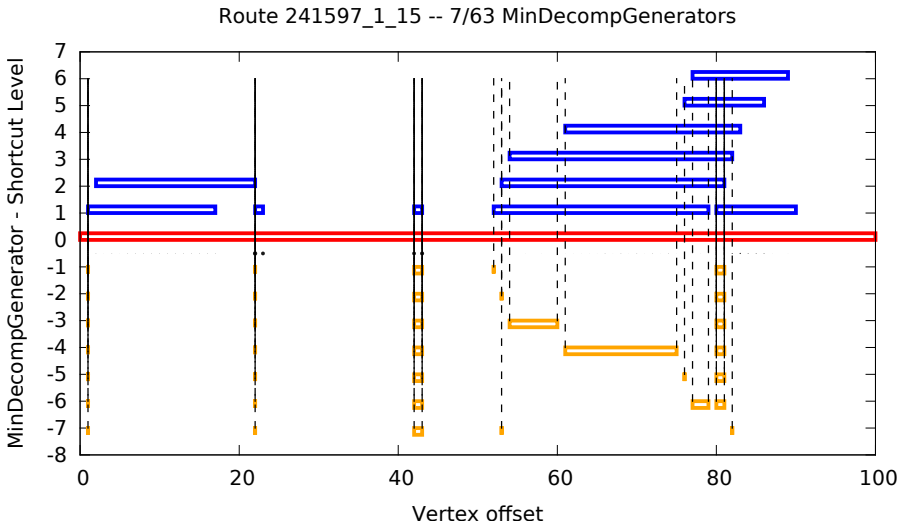


Figure 8: The legend is explained in the caption of Figure 6. Clique number = 6. The orange layers represent 7 out of 63 MDG

Figure 9 shows the *normalized path family importance* determined over the

set of the 99147 routes processed by the machine sd-124739. The diameter of the circle at each network node is proportional to the normalized path family importance value. It is clear that vertices in the network can easily be distinguished using their importance value.

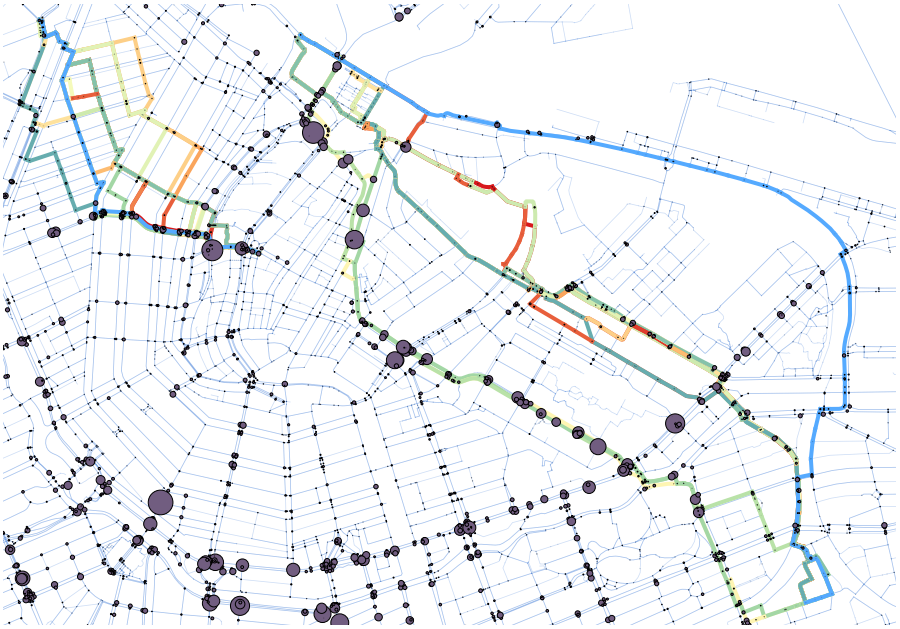


Figure 9: Normalized path family based importance for network nodes. The diameter of the circle at the node is proportional to the path family based importance value.

5. Discussion - Conclusion

A new technique to enumerate all decompositions of a given path in a graph is developed. It consists of several steps. For one of these steps, a novel efficient algorithm is presented to enumerate all minimum sequential clique covers of an indifference graph.

The concept of vertex importance is defined as relative occurrence frequency of a vertex as a split vertex in minimum path decompositions. An efficient technique is provided to compute vertex importance without brute force enumeration of minimum path decompositions.

The proposed techniques were implemented and applied to 500k predicted bikers routes for Amsterdam. The experiment shows that large datasets can be processed in (between 1 and 2 routes per second) and that normalized importance values clearly identify network nodes as intermediate destinations.

6. Future Research

Currently ongoing research focuses on the validation and enhancement of route choice sets using the distribution for path complexity. A high quality dataset to accomplish this came recently available.

The same dataset is sufficiently large to focus on the interpretation of vertex importance values in terms of land use and road network properties. We would like to validate our formulas for path based importance of a vertex, as well as normalized path family importance of a vertex on real data.

Acknowledgement

The authors thank Lara Montini (ETH Zürich - D-BAUG - IVT) for the support related to operating the `POSDAP` software.

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

[1] Knapen L, Hartman IBA, Schulz D, Bellemans T, Janssens D, Wets G. Determining structural route components from GPS traces. Transportation Research Part B: Methodological 2016;90:156–71. URL: `http://www.sciencedirect.com/science/article/pii/S0191261516302296`. doi:`10.1016/j.trb.2016.04.019`.

[2] Bovy PHL. On Modelling Route Choice Sets in Transportation Networks: A Synthesis. Transport Reviews 2009;29(1):43–68,. doi:`10.1080/01441640802078673`.

[3] Prato CG. Route choice modeling: past, present and future research directions. Journal of Choice Modelling 2009;2(1):65 – 100. URL: http://www.sciencedirect.com/science/article/pii/S1755534513700058. doi:10.1016/S1755-5345(13)70005-8.

[4] Prato CG. Meta-analysis of choice set generation effects on route choice model estimates and predictions. Transport 2012;27(3):286–98. doi:10.3846/16484142.2012.719840.

[5] Prato CG, Bekhor S. Applying Branch-and-Bound Technique to Route Choice Set Generation. Transportation Research Record 2006;(1985):19–28. doi:10.3141/1985-03.

[6] Prato CG, Bekhor S. Modeling Route Choice Behavior: How Relevant Is the Composition of Choice Set? TRB Research Record 2007;2003:64–73. doi:10.3141/2003-09.

[7] Frejinger E. Random sampling of alternatives in a route choice context. Transport and Mobility Laboratory (TRANSP-OR), EPFL 2007;.

[8] Rieser-Schüssler N, Balmer M, Axhausen KW. Route choice sets for very high-resolution data. Working Paper ETH Zürich eth-5386; ETH Zürich; Zürich; 2012.

[9] Vacca A, Prato CG, Meloni I. Estimating Route Choice Models from Stochastically Generated Choice Sets on Large-Scale Networks. TRB Research Record 2015;2493:11–8. doi:10.3141/2493-02.

[10] Kaplan S, Prato CG. Joint modeling of constrained path enumeration and path choice behavior: a semi-compensatory approach. In: Proceedings of the European Transport Conference. Association for European Transport; 2010,.

[11] Schüssler N, Balmer M, Axhausen KW. Route Choice Sets for Very High-Resolution Data. In: TRB 2010 Annual Meeting. Washington, DC, USA: TRB (Transportation Research Board); 2010, p. 16.

[12] Halldórsdóttir K, Rieser-Schüssler N, Axhausen KW, Nielsen OA, Prato CG. Efficiency of choice set generation methods for bicycle routes. EJTIR European Journal of Transport and Infrastructure Research 2014;14(4):332–48.

[13] Fosgerau M, Frejinger E, Karlstrom A. A link based network route choice model with unrestricted choice set. Transportation Research Part B 2013;56:70–80. doi:10.1016/j.trb.2013.07.012.

[14] Mai T, Fosgerau M, Frejinger E. A nested recursive logit model for route choice analysis. Transportation Research Part B: Methodological 2015;75(Supplement C):100 –12. URL: http://www.sciencedirect.com/science/article/pii/S0191261515000582. doi:10.1016/j.trb.2015.03.015.

[15] Kazagli E, Bierlaire M. Revisiting Route Choice Modeling: A Multi-Level Modeling Framework for Route Choice Behavior. In: STRC 2014. Ascona; 2014,.

[16] Kazagli E, Bierlaire M, Flötteröd G. Revisiting the route choice problem: A modeling framework based on mental representations. Journal of Choice Modelling 2016;19:1 – 23. URL: http://www.sciencedirect.com/science/article/pii/S1755534515300518. doi:10.1016/j.jocm.2016.06.001.

[17] Knapen L. Refined tools for micro-modeling in transportation research. Doctoral Thesis; Hasselt University; Diepenbeek, Belgium; 2015. URL: http://hdl.handle.net/1942/19732.

[18] Hartman Ben-Arroyo I, Knapen L, Bellemans T. Enumerating minimum path decompositions to support route choice set generation. Procedia Computer Science 2017;109(Supplement C):196 – 203. URL: http://www.sciencedirect.com/science/article/pii/S1877050917309894. doi:10.1016/j.procs.2017.05.325.

[19] Bondy J, Murty U. Graph Theory; vol. 244 of *Graduate texts in Mathematics*. Springer; 2008. ISBN 978-1-84628-969-9. Doi:10.1007/978-1-84628-970-5.

[20] Dijkstra E. A note on two problems in connexion with graphs. Numerische Mathematik 1959;1(1):269–71. URL: http://dx.doi.org/10.1007/BF01386390. doi:10.1007/BF01386390.

[21] Golumbic MC. Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57). Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co.; 2004. ISBN 0444515305.

[22] Montini L, Schüssler N. Position Data Processing. 2015. URL: https://sourceforge.net/projects/posdap/.