5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014)

Canonic Route Splitting

Luk Knapena,∗, Tom Bellemansa, Davy Janssensa, Geert Wetsa

aHasselt University, Wetenschapspark 5, B3950 Diepenbeek (Belgium)

Abstract

There are multiple ways to split a path in a directed graph into largest sub-paths of minimal cost. All possible splits constitute path partitions of the same size. By calculating two specific path splittings, it is possible to identify subsets of the vertices (splitVertexSets) that can be used to generate every possible path splitting by taking one vertex from each such subset and connecting the resulting vertices by a least cost path. This is interesting in transportation science when investigating the hypothesis that people build up their route from least cost components. The splitVertexSets can be easily and efficiently derived from big data (GPS recordings). This allows for statistical analysis of structural route characteristics which in turn can support constrained enumeration methods for route choice set building. Furthermore, the boundary vertices separating consecutive route parts, are way points having a particular meaning to their user which constitutes relevant information to the transportation analyst.

1. Research context · Objectives · Related work

Travel demand prediction by means of microsimulation in activity based models, results in an agenda for each individual for the simulated period of time. Such agenda consists of a sequence of episodes each one of which is defined by a period of time, an activity type, a location and the mode used to reach the location. As soon as the locations are known, the traffic demand needs to be assigned to the transportation network. Thereto route choice sets and route choice models are required.

Route choice procedures in general consist of two parts: a route choice set generator and a route choice model. Route choice is based on generalized cost; the driver selects an optimal route having limited information and limited processing capacity. Bekhor1 presents an overview of route choice procedures. The paper discusses several methods for route choice set generation and introduces the concept of coverage to quantify their quality. Routes driven by 188 respondents in an experiment have been recorded and used to calculate the choice sets coverage. MNL (Multinomial Logit), PSL (Path Size Logit) and two CNL (Cross-nested Logit) models are estimated and compared.

Frejinger2 constructs the route choice set by sampling from the universal set of all paths from origin to destination. Routes are constructed by assigning a weight to each link in the network based on its distance to the shortest path. Then a random walk is started at the origin; in each node the algorithm selects the next link using importance sampling.

∗ Corresponding author: Tel.: +32-11-269-126; Fax: +32-11-269-199
E-mail address: luk.knapen@uhasselt.be
based on the link weights until the destination is reached. The path size correction attribute in the route choice step of PSL, then is expanded to take into account correlation with paths that have not been sampled as members of the choice set.

In the authors introduce the subnetwork concept. A subnetwork is a set of links that can easily be labeled and is behaviorally meaningful. Each subnetwork is to be defined by the analyst (possibly by interviews). Paths sharing subnetworks are correlated and the correlation is accounted for in the route choice phase (the second step in route generation).

According to most route choice models relate to revealed choice behavior. The author presents branch-and-bound techniques that construct a connection tree between origin and destination by processing sequences of links according to a branching rule that accounts for behavioral constraints and has been formulated to increase route likelihood. The paper compares several generation techniques and reports that coverage levels attained by branch-and-bound techniques are much higher than for other techniques.

In Prato states that the use of a well defined route choice set results in more realistic routes than purely link based procedures. The author presents a comprehensive overview of route choice set generation techniques and recognizes the difficulties with path enumeration and with the requirement to add all relevant paths to the choice set while it is very difficult to define route relevancy.

Fosgerau distinguishes between three approaches for route choice modeling: (i) Path based models that require a route choice set from which routes will be selected using a route choice model like PSL (Path Size Logit), C-Logit (commonality factor), CNL (Cross-nested Logit) etc. The route choice set consists of observed routes and of routes sampled using a path generator algorithm. (ii) the path based model described in that assumes that the route choice set contains all feasible paths and path utility is corrected for the sampling protocol used and (iii) the recursive link based method. This method calculates the expected path utility in a dynamic programming context by including the expected path utility in the Bellman value function. The author shows how to estimate the parameters by maximum likelihood estimation. This technique avoids the use of a route choice set.

This paper aims to contribute to route choice set generation. It belongs to the category of constrained enumeration methods mentioned in. We investigate the hypothesis that for utilitarian trips, individuals tend to construct their route as a concatenation of a low number of minimal cost routes. The individual is assumed to make use of some preferential locations between their origin and destination and to travel in the most efficient way between those intermediate locations. During the route choice set generation, realistic routes shall be generated. The number of shortest sub-paths shall be realistic. We aim to investigate the characteristics of a large set of GPS-recorded and map matched routes. We are interested in the distribution for the number of shortest paths in each route. Furthermore, it is not known in advance why particular intermediate locations in non-shortest paths are chosen. Therefore, we want to analyze the use frequency of nodes as split nodes in the routes in order to verify the hypothesis that some nodes are preferential trip splitters due to traffic related characteristics like availability of traffic lights. The results of our research can be used in branch-and-bound rules mentioned in to partially replace threshold values based on expert opinion. Split node sets (see section 2.2) could serve to create the subnetworks mentioned in. This will integrate revealed evidence from big data in the choice set generation phase. Moreover, not only scalar path characteristics like the ratio of the actual to shortest path length, but also structural route characteristics are used.

The research described in this paper focuses on the splitting of routes into basic components. The term route splitting has been deliberately chosen since path decomposition and path partitioning denote problems in graph theory different from what is discussed in this paper. The terms node, link and route are used in the context of road networks; the terms vertex, edge, path are used when the corresponding graph context is used. The terms shortest path and length are avoided: path or route size is used to denote the number of vertices (nodes); the term cost is used to denote the effort required to cross an edge (link).

Section 2 defines the concept and explains the algorithm used. Section 3 describes an experiment conducted on a set of traces; finally, section 4 explains future research plans.
2. Canonic splitting of a path in a graph

2.1. Definitions

**Definition 2.1** (Split). A Split of a path \( P \) in a graph \( G \) is a partition of the ordered set of edges constituting the path \( P \).

**Definition 2.2** (SplitVertex). A SplitVertex is a vertex that does not belong to two edges in the same part of the split. A split vertex either belongs to a single edge (and corresponds to an endNode on the route) or belongs to vertices belonging to different parts.

**Definition 2.3** (BasicPathComponent). A Basic Path Component is a contiguous part of a path that is either a least cost path or a single edge that does not constitute the least cost path between its vertices.

As a consequence, a SplitVertex is either a boundary vertex between two consecutive BasicPathComponents or an endVertex of the path.

**Definition 2.4** (CanonicSplit). A Canonic Split of a path is a split having a minimal number of parts and for which each part constitutes a BasicPathComponent

2.2. Characteristics of canonic splits

Consider the operation \( \diamond_f \) that determines the largest head basic path component (BPC) of a path: this is the BPC of maximal size that contains the first node of the path. The operation \( \diamond_f \) splits the path \( P \) in a head part \( H \) and a tail part \( T \). Let \( H\diamond_f(P) \) denote the operation that delivers the head of \( P \) by applying \( \diamond_f \) and let \( T\diamond_f(P) \) denote the operation that delivers tail. Then

\[
P_0 = P \\
P_1 = T\diamond_f(P_0) = P_0 \setminus H\diamond_f(P_0) \\
P_k = T\diamond_f(P_{k-1}) = P_{k-1} \setminus H\diamond_f(P_{k-1}) \cdot \cdot \cdot H\diamond_f(P_0) \\
\]

Equation (2) shows how the operations \( T\diamond_f \) and \( H\diamond_f \) split a path \( P \). Equation (3) shows how \( P_k \) is constructed by stripping of a head BPC \( k \) times. If \( P_m = \emptyset \), then the sequence \( T\diamond_f(P_0), \ldots, T\diamond_f(P_{m-1}) \) constitutes a canonic split of \( P \).

In a directed graph the cost for the vertex pair \((V_0, V_1)\) in general differs from the cost for the pair \((V_0, V_1)\). The cost to traverse an edge shall be non-negative. The path is traversed in the forward direction i.e. neighbors are handled in the order \( V, succ_p(V) \). When applying \( \diamond_f \), the cost associated with the pair for which the order is compatible with the path traversal order i.e. \((V, succ_p(V))\), is considered.

Then consider the operation \( \diamond_b \) that is similar to \( \diamond_f \) but starts at the last vertex in the path and uses backward path traversal. Neighbor nodes now are handled in the order \( V, pred_p(V) \) but the cost used by \( \diamond_b \) also is the one associated with \((V, succ_p(V))\).

Please refer to figure 1 while reading theorem 2.1 below. Each labeled square at the bottom (A . . . J), each corresponds to a vertex in path \( P \); \( A \) corresponds to the first (origin) vertex and \( J \) corresponds to the last (destination) vertex. Each small circle in the triangular structure, corresponds to a subpath of \( P \) joining the vertices that are reached from the circle by following the diagonal lines. Hence the circles at the \( i \)-th layer correspond to subpaths of size \( i \) etc.

Each non-white circle corresponds to a subpath that is a least cost path in the graph. The black circles correspond to least cost subpaths of largest size. Since each subpath of a least cost path is also a least cost path (the inverse is not necessarily true) all circles in a triangle having a black circle a the top, correspond to a minimal cost path (those are the gray circles). The non-shaded (white) circles correspond to non minimal cost subpaths.

The minimal cost subpaths are constructed by repeated application of \( \diamond_f \) The process of subpath construction by \( \diamond_f \) is visualized by the straight continuous line arrows in figure 1. From the first vertex, the maximal size least cost path is found by following the colored circle at the highest level (arrow from vertex \( A \) to circle labeled \( AD \)). The process is repeated from the last vertex in the subpath: this is found by following the downward pointing straight arrow. The resulting subpaths in the figure are: \( AD, DH \) and \( HJ \).
Fig. 1. Minimal cost paths hierarchy. The squares at the bottom of the triangle represent vertices consisting a path in a graph. The circles represent subpaths. The subpath from $V_0$ to $V_1$ is represented by the top of the triangle having $V_0$ and $V_1$ as its base vertices. Shaded circles correspond to least cost subpaths. The black circles correspond to maximal size least cost paths.

Fig. 2. Numbering of basic path components and split vertices in a decomposition.

<table>
<thead>
<tr>
<th>1</th>
<th>k</th>
<th>k+1</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>k</td>
<td>k+1</td>
<td>N</td>
</tr>
</tbody>
</table>

Component numbers

SplitVertex (set) numbers

The process of subpath construction by $\diamond_b$ is visualized by the curved dashed line arrows in figure 1. The same method now results in the path subpaths $AD$, $DG$ and $GJ$. All proofs have been left out due to lack of space.

**Theorem 2.1.** The number of BPC found by $\diamond_f$ is identical to the one found by $\diamond_b$.

The reasoning in the proof corresponds to observation that the limiting vertices of a BPC cannot belong to the same shaded triangle because in that case the BPC would not have maximal size. This holds for every BPC, so the endpoints of the $BPC^f$ form a weakly alternating sequence with those of the $BPC^b$. Weak alternation means that some vertices can coincide (like D in the figure 1).

**Theorem 2.2.** The number of BPC in the result of $\diamond_f$ is minimal.

In similar way, the number of BPC in the result of $\diamond_b$ is minimal too. For basicPathComponent and splitVertex numbering, see figure 2: the first node on the path gets number 0, the BPC starting at node $k-1$ gets number $k$. Nodes and BPC are numbered consecutively in the order in which they appear in the path.

**Definition 2.5 (SplitVertexSet).** The $k$-th SplitVertexSet of a path is the set of vertices that occur as the boundary vertex between the $k$-th and the $(k+1)$-th BasicPathComponent in at least one canonic split.

Examples of split vertex sets in figure 1 are $\{D\}$ and $\{G, H\}$.

**Lemma 2.1.** SplitVertexSets constitute a subpath of the decomposed path $P$ (i.e. they consist of contiguous vertices in $P$).

**Theorem 2.3.** The SplitVertexSets for given path $P$ are mutually disjoint.

Since the subpath joining an arbitrarily chosen vertex from splitVertexSet $R_k$ to an arbitrarily chosen vertex in splitVertexSet $S_{k+1}$ is a least cost path and since the selections from both splitVertexSets are mutually independent
(since the sets are disjoint), the total number of canonic splits for path $P$ is given by

$$N_P = \prod_{i \in [1, |S| - 1]} |S_i| \quad (4)$$

2.3. Splitter Algorithm

The headHunter algorithm is given an origin vertex on the path and finds the maximal size basic path component starting at the given vertex. It starts at the given origin with a subpath consisting of one edge, checks whether the subpath constitutes a basicPathComponent and, if so, extends the subpath with the next edge (if any); this is done until the extended subpath would no longer be a basicPathComponent. The headHunter algorithm is based on the Dijkstra least cost path algorithm. For a given path $P$, it needs to calculate the least cost path from a given origin $V_O \in \text{vertices}(P)$ to every vertex on $P$ consecutively. Consider a vertex $V_k \in \text{vertices}(P)$. In order to minimize the computational effort when calculating the least cost path from $V_O$ to $V_{k+1} = \text{succ}(V_k)$, part of the Dijkstra queue is reused. After executing the Dijkstra algorithm, the vertices belong to 3 classes: (i) unVisited (have not been visited), (ii) final (have been popped from the queue), (iii) unfinished (are on the queue). For the calculation of the least cost path to the next vertex, (i) the queue is initialized with all final nodes having at least one non-final neighbor and (ii) all unfinished nodes are reset to unvisited. The algorithm is not show due to lack of space.

2.4. Application

When splitting traveled routes, the splitVertexSets can easily be determined. However it is not known which of the possible canonic splits the user had in mind while traveling: i.o.w the researcher cannot derive solely from the decomposition which one of the route nodes corresponding to the splitVertices were relevant to the traveler. On the other hand, the splitVertexSets deliver minimal sets of vertices and hence minimal sets of route nodes to be investigated by verifying the transportation related characteristics.

3. Experiment performed

Data recorded by GPS receivers need to be map matched onto a network in which each link and node is attributed with a cost. The map matching step is crucial. Some map matchers try to fill (small) gaps in the recording by assuming that the traveler moved along a least cost path (according to some criterion). This shall not be done in this research because the hypothesis to be tested shall not be influenced by hypotheses used while map matching.

Real data from recorded GPS traces are not available at the moment of writing but are expected soon. Hence, an initial experiment to evaluate the canonic path splitter has been conducted using synthetic traces. The generation process is described in the next subsection.

3.1. Dataset used for splitting

OpenStreetMap (OSM) was used to extract a road network for Flanders (Belgium). The network has 479920 links and 372608 nodes. The links between network nodes in general are not straight lines. The direction of a link is by convention defined using the coordinates of its endpoints. The link length however, is taken from the OpenStreetMap database and hence is the developed distance along the geometric trail associated with the link.

2000 routes have been generated. The route origin was selected by sampling from the network nodeset using a uniform distribution. For each route, the bearing was sampled from uniform(0, 2\pi) and the route length from uniform(1, 120) [km]. The distribution for the path sizes (number of links) is shown in figure 3. The average number of links in a route is 238. Hence the average number of least cost route determinations was 237 per route.

The traveler starts at the origin node. In each node the outgoing links leading to a node not yet belonging to the path (if any) are considered. For each of them, the angular difference $\alpha$ between the conventional link direction and the given bearing is calculated. Finally, the outgoing links are sorted to decreasing cos($\alpha$) values. The generator recursively tries to add a link to the route being built; links having larger cos($\alpha$) are used first and each node shall be visited at most once. The algorithm stops if the cumulative length of all added links, exceeds the required length. The
traveler hence is assumed to use a compass and to select the best conventional direction in each node. Figure 4 shows some generated examples.

Note that the network is bounded since only the region of Flanders was used. When the location determined by the origin, the bearing and the straight distance is outside the bounded map, the algorithm will create a route that remains near the border and keeps trying to leave the region: an example is shown in figure 5. Routes for which this phenomenon occurs, have a large number of basic route components. Hence the statistics drawn from this set are not expected to mimic those for the real world traces.

3.2. Results

Routes had been generated for a non cleaned network that contained parallel links having identical lengths. Those have been dropped afterwards so that while loading the routes for processing, some had to be dropped due to missing links. Decomposition was done for 1684 routes. Total calculation time was 11300[sec] or 6.7[sec] per route (0.028[sec] / leastCostPath for the approximately 1684*237=399108 cases).

| Average number of links per basicRouteComponent | 11.87 |
| Average straight distance origin to destination / developed length | 0.62 |

4. Future research

In the current version of the canonic route splitting software, some nodes are repeatedly moved between unfinished and unvisited states. It is to be found out whether or not this can be avoided by replacing the adapted Dijkstra algorithm by an adapted A* algorithm.

Map-matched GPS recordings will become available soon. Application of the algorithm to those data is expected to show less basicPathComponents per route than the for the synthetic routes because real travelers pursue a goal (the (intermediate) destination) whereas in the synthetic routes the virtual traveler in each node selects the link based on the specified bearing.

SplitVertexSets will first be analyzed in a GIS in order to find hints for use patterns. Finally, a method to generate routes between specified origin and destination based on the frequency distribution for the number of components, is to be elaborated.
Fig. 4. Sample of generated routes that did not suffer from redirection by border limits. The horizontal distance is about 58[km], the vertical distance 45[km]. The cities shown are Antwerp (top left) and Brussels (bottom left).

Fig. 5. Both the (partially overlapping) red and blue routes start at their upper right location and try to move to the lower left. When stuck at the border, they continue to add road segments according to the specified bearing (south-west direction) until the required route length has been reached. The change in direction near the beginning of the blue route is caused by recursion while trying to find a sufficiently long route.
5. Conclusion

Route characteristics are required to for route choice set generators used in traffic demand simulation. Canonic splitting of paths in a graph has been developed. It constitutes an unambiguous way to characterize paths and allows to mathematically determine sets of visited points that need application level investigation because they are boundary nodes between least cost route components.

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 270833.

References