Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 109C (2017) 196–203

Procedia Computer Science

www.elsevier.com/locate/procedia

The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017)

Enumerating minimum path decompositions to support route choice set generation

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

[a]Caesarea Rothschild Institute for Interdisciplinary Applications of Computer Science, University of Haifa, Haifa 31905, Israel
[b]Transportation Research Institute (IMOB) Hasselt University, Diepenbeek, Belgium

Abstract

This paper concerns the structure of movements as were recorded by GPS traces and converted to routes by map matching. Each route in a transportation network corresponds to a collection of directed paths or cycles in a digraph. When considering only directed paths, corresponding to utilitarian trips, the path is not necessarily a shortest path between its origin and destination, and can be split up into a small number of segments, each of which is a shortest or least cost path. Two consecutive segments are separated by *split vertices*. Split vertices act as intermediate destinations in the mind of travellers who try to hop between them using minimum cost paths. Hence they provide useful information to build route choice models. In this paper we identify and enumerate all possible decompositions of a path into a minimum number of shortest segments. This gives us an indication of the importance of split vertices occurring in particular sets of revealed routes that belong either to a single traveller or to a specific group. The proposed technique allows for automatic extraction of frequently used intermediate destinations (way-points) from *revealed preference* data.

Keywords: clique cover, proper interval graph, indifference graph, transportation network, route choice set generation, GPS traces

1. Introduction

We begin with a short background from transportation science and graph theory in order to motivate our problem. We model a transportation map by a directed graph. where the vertices denote junctions or points of interest such as a petrol station, shop, restaurant, rest area, or any point where drivers may choose to stop. Each edge of the graph corresponds to a road segment and represents a set of lanes having the same direction (forward or backward). Travelers move between locations on the *geometries* of source and destination segments. Transportation scientists are interested in modelling route choice behaviour in order to forecast and simulate travellers' decisions under different conditions and resources of information. See Bovy[3] for a review on route choice set generation and selection. The *branch and bound* technique by Prato[11,8] generates candidate routes to populate the choice set and removes the ones that do

* Corresponding author. Tel.: +972-4-8288370 ;
E-mail address: irith.hartman@gmail.com

not meet specific constraints (e.g. number of traffic lights, left turns, and others). We proposed in[10] an additional criterion for route choice set, which is the minimum number of shortest subroutes which constitute a given route. In[10] 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 rather a route which is a concatenation of a *small* number of shortest (cheapest) routes. The vertices connecting these shortest routes may have a special significance for the traveller, they are intermediate destinations which can be quantified by the use frequency of these vertices in the set of all paths chosen by the travellers. The intermediate destinations support the route choice modeling. Finding these vertices, and all possible ways of breaking up a path into a minimum number of shortest paths relates to the problem of finding some minimum clique covers in an indifference graph - as will be shown in the following sections.

2. Definitions and Basics

We begin with some basic standard definitions from graph-theory. We use definitions and notations as in[1]. 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. For a subgraph $H \subseteq G$, let $V(H)$ and $E(H)$ denote the sets of vertices and edges of H, respectively.

A *walk* is a sequence of vertices $P = (v_0, v_1, \ldots, v_l)$, not necessarily distinct, where $(v_i, v_{i+1}) \in E(G)$ for all $i = 0, 1, \ldots, l-1$. Vertices v_0 and v_l are called *initial* and *terminal* vertices, respectively, of P, and vertices $v_1, \ldots, v_{l-1}$ are called *internal vertices* of P. The walk P is said to be *connecting* v_0 and v_l, and it is also denoted by $P(v_0, v_l)$. A walk $Q(v_0, v_l)$, is *internally-disjoint* from P if all the internal vertices of Q, are distinct from the vertices in P.

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 $0 \le i \le j \le l$ 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[10].

Definition 2.1 (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 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.

Clearly, a least cost path cannot have any shortcuts.

Definition 2.2 (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, and is denoted by v_i^s.*

We remark that there may be many ways to split a path, for example, the trivial partition into 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

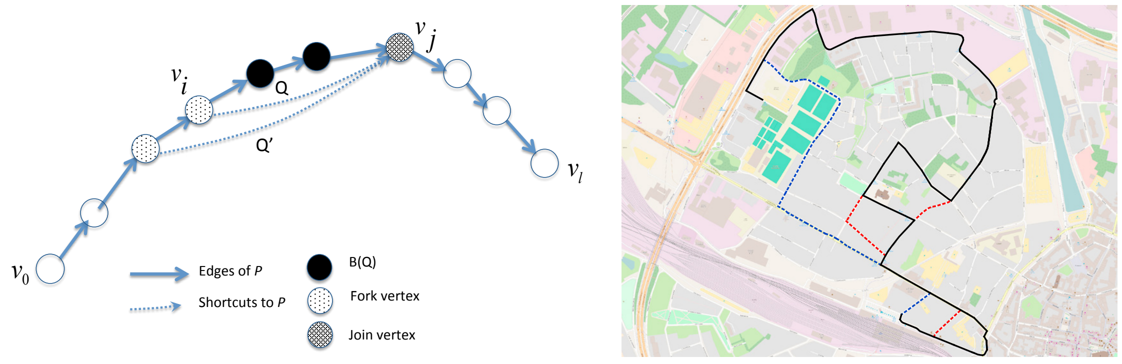

Fig. 1: Shortcuts: (a) A path P with a minimal $P(v_i, v_j)$-shortcut Q. Q' is a non-minimal shortcut. Vertices v_i and v_j are fork and join vertices of Q, respectively, and the dark vertices are the bypassed vertex set $B(Q)$. (b) A path in the road network. The dashed lines represent shortcuts (minimal in red, non-minimal in blue)

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. 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.

From now on we will assume that P does not contain any non-shortest-edges.

In[10] 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.

Lemma 2.1. *Let $P = (v_0, v_1, \ldots, v_l)$ be a path connecting v_0 and v_l. Assume P is not least cost, and let $Q(v_i, v_j)$ be a shortcut in P. Then any path splitting of P will contain at least one vertex in $B(Q)$, the Q-bypassed vertex set, as a splitVertex.*

Proof: By contradiction. If no vertex in $B(Q)$ is a splitVertex, then $P(v_i, v_j)$ is a least cost path, contrary to the fact that $Q(v_i, v_j)$ is a shortcut in P. ■

Corollary 2.2. *A minimum path splitting of P is obtained by a minimum set of splitVertices which intersects $B(Q)$ for all minimal shortcuts Q to P.*

Proof: Since every two consecutive BPC are separated by a splitVertex, it is sufficient to minimize the number of splitVertices. By Lemma 2.1 it is necessary to meet each $B(Q)$, where Q is a shortcut. However, since every $B(Q')$ contains a $B(Q)$ where Q is a minimal shortcut, it is sufficient to meet all bypassed sets of minimal shortcuts. The converse also holds - a minimum set of vertices which intersects the $B(Q)$ of all minimal shortcuts Q, defines a minimum path splitting of P. ■

The algorithm described in[10] begins with the initial vertex of P, v_0, and finds a *maximal* least cost path beginning with it, i.e. a path which cannot be extended without repeating vertices. This is done using Dijkstra's[4] least cost path algorithm. Assume v_{j_1} is the first vertex on P for which $P(v_0, v_{j_1})$ is not least cost, then the algorithm marks v_{j_1} as a join vertex and continues with the subpath of P beginning from the vertex prior to v_{j_1} on P, looking for the next join vertex in $P(v_{j_1-1}, v_l)$. The algorithm continues until no more join vertices are found. It was proved that the vertices preceding the join vertices found on P are splitVertices, and their number is minimal.

In a similar way, a backward pass on P is done, beginning with the end vertex v_l and going backwards, to find a minimum set of fork vertices, whose successors on P are also splitVertices in a minimum path splitting. The algorithm

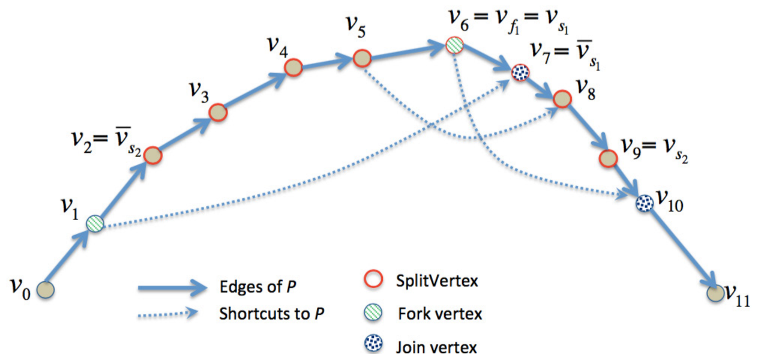

Fig. 2

is efficient since it uses Dijkstra's algorithm no more than N times, where N is the minimum number of BPC's used to partition P.

Since the bypassed vertex sets may contain points of interest for the traveller (otherwise a minimum cost path would have been chosen), we are interested in enumerating all splitVertices and all minimum partitions into BPC's.

We remark that the algorithm in[10] is highly efficient, but it does not find all the shortcuts to P. This fact does not allow us to enumerate all possible minimum path splittings, as was demonstrated in the example in[10], Figure 3. We depict this example here, for the sake of completeness. In Figure 2 the shortcut from v_5 to v_8 is not found using the algorithm described above.

3. Path Decomposition Enumeration Technique

Assume that a path P in a graph is given. Without loss of generality we assume it is free of non-least-cost edges. The enumeration of all minimum partitions of P into BPC's is done in three steps (see also[9]):

1. Finding all minimal shortcuts to P.
2. Defining the intervals and the proper interval graph G^I derived from the shortcuts to P.
3. Enumerating all minimum decompositions of P
   (a) Finding all sequential cliques in G^I
   (b) Constructing the directed sequential -clique-graph G^C
   (c) Enumerating all shortest source-sink paths in G^C
   (d) Enumerating all minimum decompositions of P

3.1. Finding all minimal shortcuts to P

In this stage we find all minimal shortcuts to P. 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.

Without loss of generality, we assume a traveller moves from point v_0 to point v_l along a path $P = (v_0, v_1, \ldots, v_l)$. If this path is the least-cost path between v_0 to v_l then it has no shortcuts, it is a BPC, and we have nothing to do. Otherwise, we use Dijkstra's[4] shortest path algorithm to find the first vertex on P, say v_j for which $P(v_0, v_j)$ is not the shortest path connecting v_0 and v_j. We mark v_j as a join vertex, and continue, by applying a backward search starting at v_j, to find 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 process ends when we reach v_l, the terminal vertex of P. (See Figure 3-Stage 1).

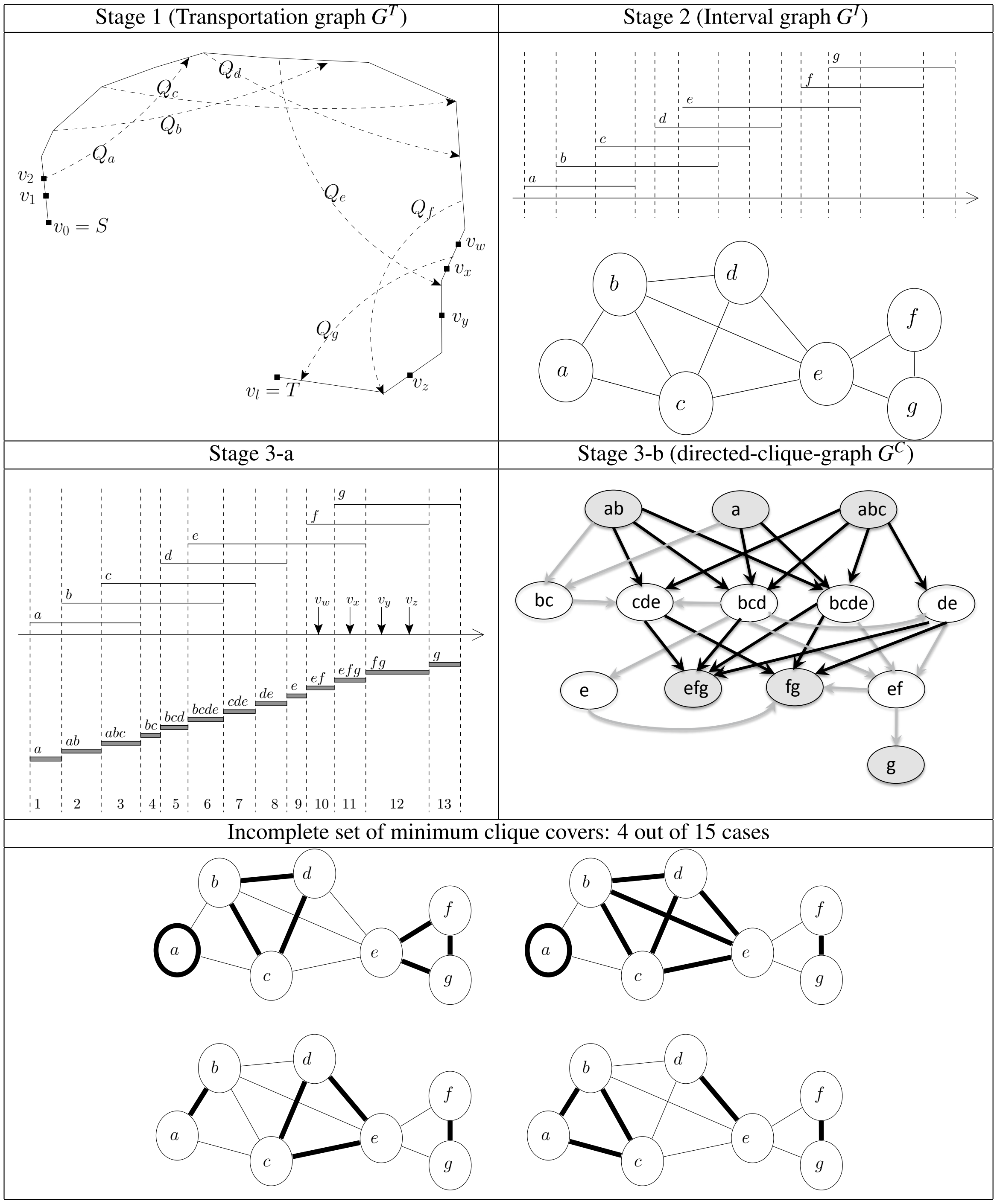
Stage 1 (Transportation graph G^T)
Stage 2 (Interval graph G^I)
Stage 3-a
Stage 3-b (directed-clique-graph G^C)
Incomplete set of minimum clique covers: 4 out of 15 cases
Q_a
Q_b
Q_c
Q_d
Q_e
Q_f
Q_g
v_2
v_1
$v_0 = S$
v_w
v_x
v_y
v_z
$v_l = T$
a
b
c
d
e
f
g
ab
abc
bc
bcd
bcde
cde
de
ef
efg
fg
1 2 3 4 5 6 7 8 9 10 11 12 13

Fig. 3: Overview of graphs used to enumerate minimum path decompositions.

3.2. *Defining the intervals and the proper interval graph G^I*

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 $v_0, v_1, v_2, \ldots, v_l$, then 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 3-Stage 2) 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[7]). 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). We label the ordered set of intervals as $a, b, c, \ldots$ (see Figure 3-Stage 2).

3.3. Enumerating all minimum decompositions of P

Let G^I be a proper interval graph and assume it is represented by intervals $a, b, c, \ldots$ ordered by their left hand endpoints. (See Figure 3-Stage 2). We remind the reader that 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.

3.3.1. Finding all sequential cliques in G^I

Assume we have a collection of intervals, as was found in section 3.2. 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. (See Figure 3-Stage 3a). (The converse is also true: every set of intervals which mutually pairwise intersect, by the Helly property, intersect in a point, or a subinterval of the real line.) Moreover, we can assume, without loss of generality, that this point is an endpoint of at least one interval. The set of all intervals contains at most $2|V^I|$ endpoints, since each interval has exactly two endpoints (the inequality comes from the fact that some right hand endpoint may coincide with a left hand endpoint). They partition the real line into at most $2|V^I| - 1$ segments which correspond to non-empty intersections of the intervals, ordered linearly in increasing order of the intersecting points (see Figure 3-Stage 3-a). (When two intervals have identical endpoints the segment will consist of a single point). Each such interval intersection is a clique in G^I, which consists of a set of consecutive intervals which mutually intersect each other, and no other intervals. We call such a clique a *sequential clique*, or for short, s-clique. For example, in Figure 3 intervals $\{b, c, d\}$ are an s-clique, but $\{c, d\}$ is a clique but not an s-clique since there are no points on the real line which meet b and d but no other interval. The s-cliques are naturally ordered so that each interval belongs to consecutive cliques, and each s-clique consists of consecutive intervals. This is a known property of proper interval graphs[2,5,6].

3.3.2. Constructing the directed-s-clique-graph G^C

Given the clique family found in 3.3.1, denoted by $C(G^I)$, we are interested in finding all possible minimum coverings of V^I by s- cliques. To do that we construct a directed graph $G^C = (V^C, E^C)$, where the vertex set corresponds to the s-cliques $C_i \in C(G^I)$. (There are at most $2|V^I| - 1$ s-cliques). We construct a directed edge from a C_i to C_j according to the following rule: Assume clique C_i contains the consecutive set of intervals labeled $I_i, I_{i+1}, \ldots, I_{i+k}$ and clique C_j contains the consecutive set of intervals labeled $I_j, I_{j+1}, \ldots, I_{j+t}$. There is a directed edge from C_i to C_j if and only if

$$i < j \leq i + k + 1 \text{ and } i + k < j + t \tag{1}$$

See example in Figure 3- Stage 3-b. By the definition above, G^C is acyclic, i.e. it contains no directed cycles. All the *source* vertices (i.e. vertices with indegree zero) are the vertices in G^C whose label contains I_1 (or a as in Figure 3-Stage 3-b) and the *sink* vertices (i.e. vertices with outdegree zero) are the vertices whose label contains I_n (or g as in Figure 3-Stage 3-b). The source and sink vertices are marked in Figure 3-Stage 3-b as dark vertices.

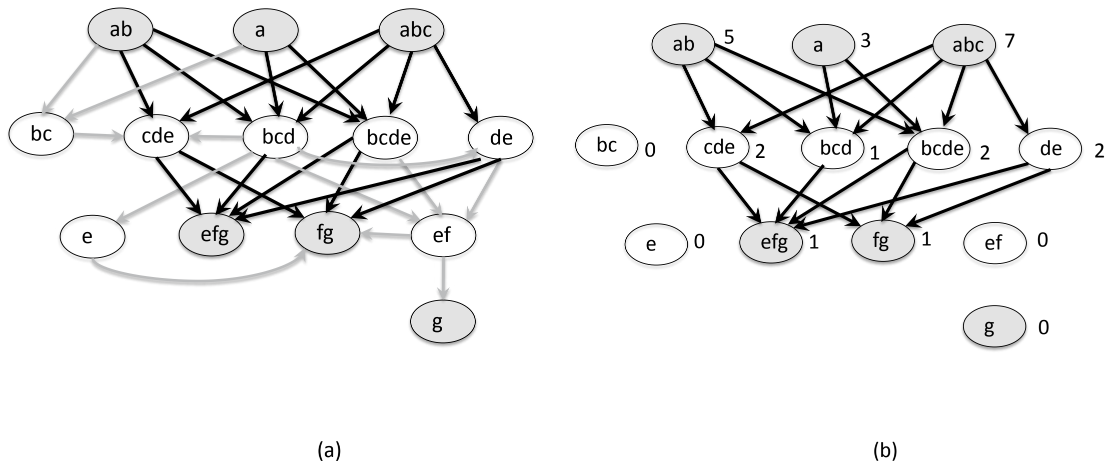

Fig. 4: The graph G^C constructed in stage 3-b: diagram (a): before pruning, diagram (b) after pruning lower layer sinks and deleting edges not on a shortest source-sink path. The numbers beside each vertex represents the number of shortest paths from the vertex to a sink vertex.

3.3.3. Enumerating all shortest source-sink paths in G^C

We look for all minimum length paths in G^C which connect between some source vertex to some sink vertex. The easiest, most efficient way to find shortest paths is to perform a breadth-first search (BFS) on G^C starting with the source vertices. Recall that BFS search divides a graph into *layers*, starting with the source vertices, in which all the nodes in layer d have distance d from the source vertices. We say that layer d is *above* layer d' if $d < d'$. Since we are looking for the shortest paths from sources to sinks, we consider only the closest sink vertices, i.e. sink vertices with the highest layers; other sink vertices we can ignore, as well as their incoming edges. We also ignore edges which connect two vertices at the same layer of the BFS tree, and edges which do not lead to any highest-level sinks. (See Figure 4).

To count the number of shortest source-sink paths, we begin with the sinks in the highest layers, and label them 1. (since there is a unique shortest path beginning at them and ending at them - the trivial path consisting of one vertex). As we move up through the BFS layers, we see that the number of shortest paths to sinks from each node is the *sum* of the number of shortest paths from all nodes directly below it to sinks in the BFS search. Working upwards through the layers, we get the number of shortest paths from each source vertex to sink vertices. It is quite easy to construct these paths layer-by-layer, as in Figure 4-(b).

3.3.4. Enumerating all minimum path decompositions

In Section 3.3.3 we have found all shortest source-sink paths in the clique graph G^C of the interval graph G^I. We recall that each vertex in such a path corresponds to a s-clique in the interval graph, in other words, to some interval of the real line, whose integer points represented a set of vertices in the original path P traveled by the user, which are bypassed by a unique set of shortcuts. If $C_{i,j}$ denotes the s-clique containing intervals $I_i, I_{i+1}, \ldots, I_j$, denote by $\mathcal{K}(C_{i,j})$ the set of integer points of the real line contained in those intervals only, and in no other intervals[1].

For example, in Figure 3 $\mathcal{K}(ef)$ 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. Each one of these bypassed vertices is a potential splitVertex and its choice is independent of the selection of bypassed vertices corresponding to the other s-cliques on the same source-sink path. (see Figure 3-Stage 3). The number of possible path splittings is therefore,

$$\sum_{P \in \mathcal{P}^C} \left(\prod_{C \in P} |\mathcal{K}(C)| \right) \tag{2}$$

[1] In[9] $\mathcal{K}(C_{i,j})$ is called the *core* of a clique

where $\mathcal{P}^C$ is the set of shortest source-sink paths in G^C and each $C \in P$ represents a s-clique in G^I corresponding to a vertex on P.

Theorem 3.1. *Stages 1-4 allow us to enumerate all minimum decompositions into BPC's of a path, and Equation 2 counts the total number of such decompositions.*

Proof: We have seen by Lemma 2.1 that any minimum path decomposition of P into BPC's requires finding a minimum set of vertices in P which meet all the bypassed vertex sets $B(Q)$. In section 3 we represented each $B(Q)$ by an interval with distinct (integer) points which correspond to the vertices in $B(Q)$. Every potential splitVertex on P belongs to a unique s-clique, so in fact we need to enumerate the number of minimum s-clique covering of the interval graph. We now claim that any directed path from a source vertex to a sink vertex in G^C corresponds to covering of the vertices of G^I by s-cliques. This follows from definition (1) of the edge set of G^C: The first part of the condition in equation (1) means that $C_i \cup C_j$ contains all intervals $I_i, \ldots, I_{j+t}$; the second part implies that $C_i \not\supset C_j$. As a consequence, C_i and C_j can be potential members of a minimum clique cover of G^I. The condition in equation (1) thus guarantees that any directed path from a source vertex to a sink vertex will cover all the intervals $\{I_i; i = 1, 2, \ldots, n\}$. Furthermore, the converse also holds: any s-clique cover of G^I will necessarily correspond to some source-sink path in G^C. The size of the source-sink path is the size of the clique cover, therefore a shortest source-sink path in G^C corresponds to a minimum clique cover of the indifference graph G^I, which corresponds to a minimum path decomposition into BPC's. ■

4. Conclusion

The path decomposition technique described in[10] finds sets of splitVertices which decompose a given path in a graph into a minimum number of *basic path components* (least cost subpaths or non-least-cost edges). Enumerating all possible decompositions, and finding in how many decompositions each splitVertex participates is more challenging, and was done efficiently in this paper. This results in a method to automatically determine the network nodes that are used as intermediate destinations. The technique is applicable to large sets of routes revealed by GPS traces; hence it allows to determine the importance of intermediate destinations by frequency counting. This in turn provides way-points for use in route choice modeling.

References

1. J.A. Bondy and U.S.R. Murty, *Graph Theory*, Graduate texts in Mathematics, vol. 244, Springer, 2008, doi:10.1007/978-1-84628-970-5.
2. K. Booth and G. Lueker, *Linear Algorithms Test to Recognize Interval Graphs and Test for the Consecutive Ones Property*, (1975).
3. P. H. L Bovy, *On Modelling Route Choice Sets in Transportation Networks: A Synthesis*, Transport Reviews **29** (2009), no. 1, 43–68,.
4. E.W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik **1** (1959), no. 1, 269–271 (English).
5. D. R. Fulkerson and O. Ross, *Incidence Matrices and Interval Graphs*, Pacific Journal of Mathematics **15** (1965), no. 3, 835–855.
6. F. Gardi, *The Roberts characterization of proper and unit interval graphs*, Discrete Mathematics **307** (2007), no. 22, 2906 – 2908.
7. M.C. Golumbic, *Algorithmic graph-theory and perfect graphs*, Graduate texts in Mathematics, ??, 2004.
8. K. Halldórsdóttir, N. Rieser-Schüssler, K. W. Axhausen, O. A. Nielsen, and C. G. Prato, *Efficiency of choice set generation methods for bicycle routes*, EJTIR European Journal of Transport and Infrastructure Research **14** (2014), no. 4, 332–348.
9. L. Knapen, *Refined tools for micro-modeling in transportation research for micro-modeling in transportation research*, Doctoral Thesis, Hasselt University, Diepenbeek, Belgium, October 2015.
10. L. Knapen, I. Ben-Arroyo Hartman, D.l Schulz, T. Bellemans, D. Janssens, and G. Wets, *Determining Structural Route Components from GPS Traces*, Transportation Research Part B Methodological (2016), no. 90, 156–171.
11. C. G. Prato and S. Bekhor, *Modeling Route Choice Behavior: How Relevant Is the Composition of Choice Set?*, TRB Research Record **2003** (2007), 64–73.