Chapter 1

FORENSIC TRACKING AND MOBILITY PREDICTION IN VEHICULAR NETWORKS

Saif Al-Kuwari and Stephen D. Wolthusen

Abstract Most contemporary tracking applications consider an online approach where the target is being tracked in real time. In criminal investigations, however, it is common that only offline tracking is possible, where tracking takes place after the fact; that is, given an incomplete trace of a suspect, the task is to reconstruct the missing parts and obtain the full trace. With the recent proliferation of modern transportation systems, target entities are likely to interact with different transportation means. Thus, in this paper, we first introduce a class of mobility models that has been especially tailored for forensic analysis then propose several instances emulating different transportation means. We then use these models to build a full-fledged offline multi-modal forensic tracking system that reconstructs an incomplete trace of a particular target. We provide theoretical evaluation of the reconstruction algorithm and show that it is both complete and optimal.

Keywords: Multimodal, trace reconstruction, mobility models, forensic tracking

1. Introduction

Traditional digital forensics was exclusively concerned with extracting evidence and traces from electronic devices that may have been associated with or used in a criminal activity. In most criminal cases, however, it would also be desirable to find additional information about particular suspects, such as their physical activities (not just the digital ones). In particular, investigating the location of suspects before, during and after a crime, may contribute significant evidence, especially if it was possible to prove that a suspect was in a particular location at a particular time that he previously denies. This kind of investigations is called *forensic tracking* [3], where tracking is conducting for forensic purposes. In most criminal cases, forensic tracking investigation is carried out in an

offline manner, where a location *trace* of a suspect is obtained and then probabilistically treated to reconstruct any missing parts; a prime example of such scenario is when a target is randomly captured by several CCTV cameras scattered over a particular area.Related work discussed how to carry out such offline forensic investigation in a vehicular setting [2]. In this paper, we extend this further and consider a multi-modal environment. In particular, we adopt a recently proposed generic trace reconstruction framework [3] and extend it to build a complete multi-modal-based forensic tracking system. We assume that we have access to location information of a target showing when and where he was observed. This will create a scattered points over an area, we then need to *connect* these points to be able to find out what routes the target could have taken. These periods of missing data (between the points in which the target was observed) are called *gaps*, which if we reconstruct properly, we can obtain the target's full trace. Since we generally need to evaluate all possible routes through these gaps in a multi-modal scenario, we will need to consider pedestrian routes, public routes, or a combination of both. Briefly, our trace reconstruction algorithm proceeds in two main phases (1) scene representation and (2) trace reconstruction. In the rest of the paper we describe these phases, but due to space restriction we had to omit some details; these can be found in the full version of the paper [1].

2. Scene Representation

In order to systematically reconstruct the target's trace, a graphical representation of the crime scene and the surrounding area has to be first generated, this process proceeds in 5 steps as follows (to simplify the notation, we will often drop unnecessary labels and tags while referring to some edges and vertices in the map):

Step 1: Map Preparation.. In this initial step, a schematic map G_M (based on an actual geographical area M) of the reconstruction scene (the area over which the target trace needs to be reconstructed) is obtained. We do not impose any restrictions on the size of G_M other than requiring it to at least cover (1) all the points at which the target was observed (the available traces of the target), and (2) the crime location(s). Formally, let $G_M = (\mathcal{V}^{G_M}, \mathcal{E}^{G_M})$ be the scene graph, where $\mathcal{V}^{G_M}$ and $\mathcal{E}^{G_M}$ are the sets of vertices and edges of G_M, then we assume that $\{X_s^{G_M} \cup C^{G_M}\} \in \mathcal{V}^{G_M}$, such that:

1 $X_s^{G_M} = \{x_1^{\kappa_p}, \ldots, x_n^{\kappa_q}\}$ is the set of all locations where the target s was observed at, where $\kappa_p < \kappa_q$ are the first and last time, respectively, s was observed in G_M,

2 $C^{G_M} = \{c_1^{\kappa_k}, \ldots, c_m^{\kappa_l}\}$ is the set of several crime locations committed between times k and l. However, to simplify the discussion, we will describe the reconstruction algorithm considering a single crime, but this is obviously applicable to multiple crimes.

Step 2: Route Marking.. In this step, relevant public transport networks (e.g. buses, trains) $B_1, B_1, \ldots, B_n \in \mathcal{B}$ are marked on G_M. A transport network $B_j \in \mathcal{B}$ consists of a set of routes $B_j = \{R_1^{B_j}, R_2^{B_j}, \ldots, R_r^{B_j}\}$, which constitute most of the vertices and edges in G_M. Since we are only marking public transport routes, vertices in a route R_i correspond to either a stop (e.g. bus/train station), denoted S-vertex, or a road turn, denoted U-vertex. Similarly, edges can either be routed (part of a route), denoted B-edge, or unrouted, denoted W-edge (the latter mostly added in Step 4). Let e^ρ be an edge of type ρ, then:

$$\rho = \begin{cases} B & \text{if } e \in \bigcup_{B_j \in \mathcal{B}} \bigcup_{i=1}^{|B_j|} R_i^{B_j} \\ W & \text{if } e \in \mathcal{E}^{G_M} \setminus \bigcup_{B_j \in \mathcal{B}} \bigcup_{i=1}^{|B_j|} R_i^{B_j}. \end{cases}$$

where the notation $|x|$ means the number of elements in the set x (the length of x), assuming that x does not have repeated elements (i.e. correspond to loop-free routes). A route R_i is, therefore, defined by the set of vertices it consists of, $V_{R_i} = \{v_1, v_2, \ldots, v_k\}$, and the edges linking these vertices $E_{R_i} = \{e_1, e_2, \ldots, e_{k-1}\}$. Once all routes are marked, we plot the available traces of the target, $X_s^M = \{x_1, x_2, \ldots, x_n\}$, which specify the times and locations where the target has been observed in G_M (these will later form the gaps in which we need to reconstruct). All x_i's are either located on top of vertices or over edges (corresponding to locations on road or at intersection), that is $x_1, x_2, \ldots, x_n \in \mathcal{V}^{G_M} \cup \mathcal{E}^{G_M}$. However, elements in X_s^M should naturally be represented as vertices, thus if any x_i is located on $e_i \in \mathcal{E}^{G_M}$, e_i is split at the location of x_i such that $e_i = e_i^1 + e_i^2$. Then, x_i is added to $\mathcal{V}^{G_M}$ (as U-vertex) and e_i is replaced by e_i^1, e_i^2 in $\mathcal{E}^{G_M}$, while updating the V^{R_i} and E^{R_i} of any route R_i passing through e_i. Next, $C^{G_M} = \{c_1, c_2, \ldots, c_m\}$, the locations of the crimes, are marked on G_M, but this time $c_1, c_2, \ldots, c_m$ may not be on top of a vertex or an edge, in which case a W-edge is created between c_i and the closest $v_i \in \mathcal{V}^{G_M}$. Notice that it is acceptable for c_i to be on top of an edge $e_i \in \mathcal{E}^{G_M}$ because C^{G_M} will not be involved in the reconstruction process. Finally, the directions of all edges $e_i \in \mathcal{E}^{G_M}$ are specified. The

directions of e_i^B (B-edges) can easily be determined by referring to their corresponding routes, while e_i^W (W-edges) are undirected.

Generally, a single edge e_i or vertex v_i cannot have two different types at the same time. If a particular vertex v_i is part of n routes $R_i, R_j, \ldots, R_n$, then it is S-vertex as long as v_i is an S-vertex in at least one of $R_i, R_j, \ldots, R_n$, otherwise it is U-vertex. In the contrary, edges are not allowed to be part of more than one route because different routes may assign different weights to their edges (see section 1.3 for how and when these weights are assigned). Thus, if there is more than one route traversing an edge, we create as many edges as there are routes. Let $e_i : v_p \to v_q$ be an edge between vertices v_p and v_q, and suppose there are n routes passing through e_i, then we relabel e_i to $e_{i,1}$ and create $n-1$ extra edges and label them $e_{i,2}, \ldots, e_{i,n}$. Thus, G_M is mixed multi-graph; that is, a graph allowing multiple directed edges to have the same head and tail vertices, and contains both directed and undirected edges.

Step 3: Vertex/Edge Labelling.. All vertices and edges (except W-edges) are assigned unique labels to specify the routes they are part of. The notation of a vertex $v_i^{\ell_i}$ with label $\ell_i = R_j^k, \ldots, R_m^n$ indicates that the i-th vertex in $\mathcal{V}^{G_M}$ is simultaneously the k-th, $\ldots$, n-th vertex of routes $R_j, \ldots, R_m$, respectively. Since all vertices are part of routes, a label ℓ should contain information about at least one route. Edges are characterised by the vertices they link, thus the notation $e_i^{\ell_i} : v_p^{\ell_p} \to v_q^{\ell_q}$ means that the i-th edge in $\mathcal{E}^{G_M}$ has head and tail at $v_p, v_q \in \mathcal{V}^{G_M}$, where $p, q \in \{1, 2, \ldots, |\mathcal{V}^{G_M}|\}$. The head v_p and tail v_q should belong to at least one common route and are ordered in succession according to the direction of the edge. If there is more than one route passing by e_i, extra parallel edges are created and labeled (Step 2).

Step 4: End Vertices.. Once all vertices and edges are labelled, a special set ρ^{G_M} is created containing all *end* vertices, these are the first and last vertices of every route $R_i \in B_j$ (the head and tail of R_i). To simplify the discussion, we will consider routes of a single transport network B_j, but this can easily be extended to multiple networks $B_m, \ldots, B_n \in \mathcal{B}$. Vertices belonging to ρ^{G_M} are found by first writing down the adjacency matrices $A^{R_1}, A^{R_2}, \ldots, A^{R_n}$ of all the routes $R_1, R_2, \ldots, R_n \in B_j$, where $|B_j| = n$ (B_j contains n routes). A particular vertex in R_i belongs to ρ^{G_M} if its corresponding row in A^{R_i} sums up to 1. We denote by $A_{i,j}^{R_y}$ the element of the i-th row and j-th column of A^{R_i}, if we need to refer for a whole row, we use the notation $A_{i,*}^{R_z}$, and similarly denote a whole

column by $A^{R_z}_{*,j}$ (i.e. $A^{R_z} = A^{R_z}_{*,*}$). Thus, formally:

$$\rho^{G_M} = \bigcup_{B_j \in \mathcal{B}} \rho^{G_M}_{B_j} = \bigcup_{B_j \in \mathcal{B}} \bigcup_{R_i \in B_j} \left(v_k : \sum_{k \in R_i} A^{R_i}_{k,*} = 1 \right)$$

Proposition 1. *A vertex $v_j \in V_i$ in the adjacency matrix A^{R_i} of a finite loop-free route (i.e. simple path) $R_i = (V_i, E_i)$, where V_i and E_i are the sets of vertices and edges forming R_i, is an end vertex if its corresponding row $A^{R_i}_{j,*}$ in A^{R_i}, sums up to 1.*

Proof. Let the route R_i be represented by the ordered sequence of vertices $v_1, v_2, \ldots, v_n$, where v_1 and v_n are the first and last vertices of R_i, these are called the end vertices. Clearly, v_1 and v_n will each be adjacent to a single vertex belonging to R_i, namely v_2 and v_{n-1}, respectively. All other vertices, $v_2, \ldots, v_{n-1}$ are adjacent to two vertices belonging to R_i; that is v_i is adjacent with v_{i-1} and v_{i+1}, for $i \in \{2, \ldots, n-1\}$. Therefore, $A^{R_i}_{1,*} = A^{R_i}_{n,*} = 1$, while $A^{R_i}_{i,*} = 2$ for $i \in \{2, 3, \ldots, n-1\}$. □

Since routes in G_M are directed, $\rho^{G_M}_{B_j} = \overrightarrow{\rho}^{G_M} \cup \overleftarrow{\rho}^{G_M}$, where $\overrightarrow{\rho}^{G_M}$ and $\overleftarrow{\rho}^{G_M}$ are the sets of head and tail end vertices of the routes in G_M.

Step 5: Additional edges.. In this final step we create additional W-edges between several vertices. A new W-edge is created between any two S-vertices if: (1) they belong to different routes, and (2) the distance between them is $\leq W_{max}$ (a particular threshold). Formally, the set η of the new W-edges is:

$$\eta = \left\{ e^{W,\ell_k}_k = v^{S,\ell_m}_m \leftrightarrow v^{S,\ell_n}_n : \ell_m \neq \ell_n \wedge d(v^{S,\ell_m}_m, v^{S,\ell_n}_n) \leq W_{max} \right\}$$

where $d(x, y)$ is the distance between x and y. Note that we here disregard the effect of the infrastructure on the W-edges, that is, we assume that there is no major obstacles between the S-edges that prevent W-edges from being created. However, integrating infrastructure information is easy since most modern maps contain such information, then we can find $d(x, y)$ by rerouting around the infrastructure and test whether it is $\leq W_{max}$. Finally, we can now formally define the graph $G_M = (\mathcal{V}^{G_M}, \mathcal{E}^{G_M})$ in terms of its edges and vertices, where $\mathcal{E}^{G_M} = E^{G_M}_R \cup E^{G_M}_W$ and $\mathcal{V}^{G_M} = X^{G_M}_s \cup C^{G_M} \cup V^{G_M}_S \cup V^{G_M}_U$.

3. Mobility Modelling

Mobility models were traditionally mainly used in computer simulation, where running an experiment (e.g. evaluating a protocol) on real

systems is both costly and inconvenient. Basically, mobility models generate artificial mobility traces that ideally resemble mobility patterns of real entities. These traces, nevertheless, cannot be directly used to reconstruct real traces that have been already made by real-life entities. This is mainly because real mobility patterns are based on human judgements, which are usually very stochastic in nature. However, we show that even though mobility traces generated by these models are sufficiently artificial, we can still use them effectively to assist our reconstruction process.

Mobility models are usually developed in a microscopic level, modelling the mobility of each object in relation to its surrounding environment and neighbouring objects, and thus generating realistically-looking traces. Most models, therefore, carefully parameterise the velocity (or rather the acceleration) and direction of the objects and repeatedly adjust them throughout the simulation. However, in our case, we only need to use the mobility models to estimate the time a target entity may have spent while moving from one point to another, completely disregarding the microscopic details; we call this class of mobility models *delay mobility models*. In addition, since we are considering a multimodal scenario (an entity occasionally changing transportation mode), we not only need to model each class of mobility, but also need to model how to *glue* different models together for a smooth transition. We now introduce several mobility delay models, and then proceed to model the transition between them.

3.1 Pedestrian Mobility Delay Model

Popular pedestrian mobility models, such as the social force model [7], cannot be directly used in our scenario because it requires detailed microscopic information which we cannot assume to possess, it also delves into details of the inter-pedestrians behaviour, which is not important in our case. We, therefore, introduce a mobility delay model, which we call PMDM, to calculate the time an entity x (pedestrian) would take to move from point a to point b (since we are here only concerned about the time). The mobility of the entity is mainly influenced by the static and moving obstacle objects which force the entity to perform a suitable *manoeuver* in order to avoid them. We represent each obstacle object (that the entity has to avoid) as a circle with known centre and radius. When manoeuvering an obstacle, the extra distance the entity has to travel is approximately the length of the arc formed by the chord cutting through the obstacle's circle based on the entity's direction. The time the entity

spends from point a to point b is then calculated as follows:

$$t^s_{a,b} = \frac{d_{a,b} + \sum_{i \in \mathcal{S}} r_i \cos^{-1}\left(\frac{2r_i^2 - c^2}{4r_i}\right) + \omega}{v_s - \epsilon} + \sum_{j \in \mathcal{M}} g((r_s + r_j) - d_{x,j}) \quad (1)$$

where $\mathcal{S}$ and $\mathcal{M}$ are the spaces of the static and moving objects, respectively (i.e. the number of such objects in the scenario), r_i is the radius of the circle surrounding an object (representing its range), c is the length of the chord cutting through the object's circle (can be obtained via secant line geometry and the direction $\overrightarrow{e_s}$ of the target), r_s is the radius of the circle surrounding the target s who moves with a speed up to v_s, $d_{s,j}$ is the distance between the centre of s and the centre of j, ω is a slight delay due to random factors imposed on the entity, such as crossing a road, ϵ is a random negative value (modelling the deceleration behaviour the entity is forced to undertake around obstacles), and $g()$ is a function defined as follows:

$$g((r_s + r_j) - d_{s,j}) = \begin{cases} 1 & \text{if } r_s + r_j \geq d_{s,j}, \\ 0 & \text{otherwise.} \end{cases}$$

which models the entity's pause time should it come across a moving object (waiting for it to move away). At first glance, equation 1 may seem to include microscopic details since it models interactions between objects, but we note that we can model these details without necessarily simulating the scenario in a microscopic level and only by assuming knowledge of the movement directions of the entity. The manoeuver behaviour of the entity around static objects (e.g. buildings) can then be easily modelled by referring to the scene map G_M, while the number of interactions between an entity and the moving objects (i.e. other pedestrians) can be estimated subjectively based on the popularity of the area and the time.

3.2 Transport Mobility Delay Models

Another class of common mobility models describes the mobility of vehicular entities, modelling public transport means, such as buses, trains, tramps, underground tubes etc. (we here do not consider private vehicles because they are irrelevant to our scenario, but they can be considered a special type of the TTMDM model below). In this class, the mobility of objects is more structured and less stochastic than those in the pedestrian models because they are usually constrained by fixed infrastructure (e.g. roadways, train tracks etc.). However, based on the infrastructure, we can easily make a distinction between two naturally different types of

vehicular mobility patterns, we call the first type *traffic-based transport* and the second *non-traffic-based transport*. Traffic-based Transport Mobility Delay Model (TTMDM) is concerned with objects whose mobility is governed by uncertain parameters that, in some cases, could affect the mobility behaviour significantly; this model describes the mobility of objects like buses, coaches and similar road-based public transports whose mobility pattern highly depends on the conditions of the road traffic, which cannot be precisely modelled in most cases. Non-traffic-based Transport Mobility Delay Model (NTMDM), on the other hand, is easier to develop because random delay factors (such as those in TTMDM) are of no or negligible effect on the entities' mobility behaviour. NTMDM is used to model the mobility of infrastructure-based public transports such as trains and underground tubes, where, apart from rare occasional signal and other minor failures, have deterministic mobility patterns.

TTMDM. Most realistic traffic-based mobility models [6] adjust the velocity of objects in such a way to avoid collisions. However, this level of microscopic modelling is not required in our scenario because we are only concerned about the time it took those objects to move from one point to another, not the actual movements they made. Thus, we consider modelling the factors that will affect this time figure, which can be estimated as follows:

$$t_{a,b} = \frac{d_{a,b}}{\bar{v}_{a,b}} + \left[D_{a,b}^{\text{traffic}} + \sum D_{a,b}^{\text{interest}} + \sum D_{a,b}^{\text{abnormal}} \right] \tag{2}$$

where $\bar{v}_{a,b}$ is the maximum allowable speed of the roadway between points a and b, $D_{a,b}^{\text{traffic}}$ is the expected traffic delay of the roadway between points a and b and can be estimated by the geographical and physical characteristics of the area as well as the current time of the day, $D_{a,b}^{\text{interest}}$ are delays incurred by point of interests (POI) located between a and b, and finally $D_{a,b}^{\text{abnormal}}$ represents abnormal events that reportedly occurred in the road segment between a and b, such as accidents, breakdowns etc., both $D_{a,b}^{\text{interest}}$ and $D_{a,b}^{\text{abnormal}}$ can be obtained offline either from public resources (e.g. maps) or from the police.

NTMDM. Modelling non-traffic-based transport is clearly more straightforward because the uncertainty of the stochastic delays the traffic-based transports suffer from is largely eliminated (or mitigated) here. This class describes the mobility of vehicular entities with fixed infrastructure, such as trains, underground tubes etc. In this case, we can calculate the time an object takes from point a to point b using the classical distance equation: $t_{a,b} = \frac{d_{a,b}}{v_{a,b}} + \sum D_{a,b}^{\text{abnormal}}$, where $v_{a,b}$ is the

fixed speed of the object on its journey from a to b spanning a distance $d_{a,b}$ (which can be obtained offline).

3.3 Multimodal Mobility Delay Model

Conventionally, when modelling the mobility of a particular object, we implicitly assume that the object's behaviour will be consistent throughout the simulation. However, in our scenario we cannot rule out the possibility that the target could have used multiple different transportation modes, each with different mobility characteristics. Thus far, we introduce three mobility delay models (PMDM, TTMDM and NTMDM) and now we model the transition between them by constructing a Multimodal Mobility Model (MMM) to assure a continuous flow of the target. Essentially, MMM will only model the *transition* behaviour between two different models (or two different carriers of the same model) because once the transition is completed, the relevant mobility model is called to simulate the mobility of the next part of the journey until another transition is required. When the transition happens between two carriers of the same mobility model (e.g. changing bus), the transition is said to be *homogenous*, otherwise if the transition happens between two carriers of different mobility models (e.g. transition from bus to train), the transition is said to be *heterogeneous*.

In a vehicular setting, we are actually interested in tracking the individual who is being transported by a carrier vehicle, not the vehicle itself, thus it is the entity who makes the transition which we need to model. Clearly, in PMDM both the entity and the carrier are a single component. When an entity shifts from any model to PMDM, the transition is smooth and incurs no delay (i.e. an individual does not have to wait before commencing a *walk* behaviour). For any other situation, however, transition modelling is required to calculate the time an entity will need to wait before shifting to the next model (or carrier). The main idea is to observe the timetables of the carriers at the transition location and calculate the transient wait time.

For level-1 transition, the entity is shifting from PMDM to either TTMDM or NTMDM, in both cases, the entity will most likely experience a slight transient delay due to the time difference of when it arrives at location x and when the next carrier belonging to the intended model stops at it. In this case, as soon as the entity arrives in x, it checks the intended carrier's timetable for the next departure time at its current location based on the current time and calculate the time difference. We will discuss this process in details when we describe level-2 transition, which can be thought of as a generalisation of level-1 transition.

Recall that in our scene graph G_M we represent roadways by edges $\mathcal{E}^{G_M}$ and intersections by vertices $\mathcal{V}^{G_M}$. The transition can only happen in an intersection, so let $v_i \in \mathcal{V}^{G_M}$ be a transition vertex which n carriers from either TTMDM or NTMDM stop at, let these carriers be denoted by $R_1, R_2, \ldots, R_n$ (this information is included in the label of v_i, see section 1.2). The first step is to obtain the timetables of these n carriers $T^{R_1}, T^{R_2}, \ldots, T^{R_n}$, and covert them into matrices $M^{R_1}, M^{R_2}, \ldots, M^{R_n}$, where the rows representing stops and the columns representing journeys. Note that the dimensions of the matrices depend on the timetables and may be different for different carriers. Next, we extract the rows corresponding to v_i from $M^{R_1}, M^{R_2}, \ldots, M^{R_n}$ and create a 3D matrix M^{v_i} by superposing these rows. The dimensions of this new 3D matrix M^{v_i} will be $1 \times L \times n$, such that $L = max\{w(M^{R_1}), w(M^{R_2}), \ldots, w(M^{R_n})\}$, where $w(M)$ is the width (number of columns) of matrix M. In other words, L is the number of journeys that are being made by carrier R_i that makes the highest number of journeys, where $i \in \{1, 2, \ldots, n\}$. Obviously, if $R_1, R_2, \ldots, R_n$ do not all make the same number of journeys, M^{v_i} will contain some undefined values. We assume access to a global clock which upon calling the procedure $cTime()$, it returns the current time. Once M^{v_i} is created, $c = cTime()$ is obtained to build a $1 \times n$ matrix $\hat{M}^{v_i} = [m_{1,1}, m_{1,2}, \ldots, m_{1,n}]$ such that:

$$m_{1,j} = \begin{cases} |c - m_{1,j,z}| + \epsilon & \text{if } z \geq c \geq z + 1, \\ \epsilon & \text{if } c = z \text{ or } c = z + 1, \\ \infty & \text{otherwise.} \end{cases}$$

where ϵ is a random delay representing the various factors that may hold the carriers off (such as traffic), plus the wait time at each stop. The matrix $\hat{M}^{v_i}$ will now indicate how long an entity at the current location x has to wait to pick any carrier $R_1, R_2, \ldots, R_n$ passing by x (regardless of whether $R_1, R_2, \ldots, R_n$ belong to the same or different model); the matrix will list all the carriers stopping at v_i along with the delay figures for each.

4. Trace Reconstruction

Classical missing data algorithms, such as EM [4] and data augmentation [8], cannot be directly used to reconstruct traces because these algorithms mainly make statistical inferences based on incomplete data, but will not reconstruct it, as required in our case. Additionally, we cannot assume that we have a sufficiently large number of available traces to be able to use these algorithms. Iterative sampling algorithms, such as Markov Chain Monte Carlo (MCMC) [5], when adapted for a missing

data setting, cannot be used here too for the same reasons. Instead, we take a different algorithmic approach to fill the *gaps* formed by the missing traces. We develop an efficient reconstruction algorithm that, using mobility delay models, selects the route(s) that the target most likely has taken through a gap given the time it spent traversing it. In the worst case scenario, the algorithm would at least eliminate several routes that the target couldn't possibly have taken, which may still make important evidence.

4.1 Trace Reconstruction

The reconstruction algorithm $\mathcal{A}_R$ first considers each gap individually, reconstructs it, repeats for all gaps, and then connects the reconstructed gaps to obtain the full trace of a target s. Abstractly, $\mathcal{A}_R$ consists of two fundamental building blocks, (1) a multi-graph traversing algorithm called Weight-Bound-Search (WBS), and (2) several mobility models. Once $\mathcal{A}_R$ is executed, it proceeds by running WBS over a gap, WBS, in turn, repeatedly calls the mobility models (possibly via $\mathcal{A}_R$) and returns a route (or routes) connecting the gap; $\mathcal{A}_P$ then reconstructs the other gaps in a similar fashion. The WBS algorithm is based on a branch-and-bound approach to optimise the reconstruction process, and uses a *crawler* for traversing the gaps to find plausible routes. For a gap $G_p : v_m \rightarrow v_n$ between vertices v_m and v_n, where $m, n \in \{1, 2, \ldots, |\mathcal{V}^{G_M}|\}$, a crawler C^{G_p} is generated at v_m and broadcasted toward v_n. The crawler C^{G_p} maintains two data structures: (1) a LIFO list of vertices and edges traversed so far $\chi_{C^{G_p}}$, and (2) a delay variable $\tau_{C^{G_p}}$. The $\chi_{C^{G_p}}$ is dynamically updated whenever C^{G_p} traverses a vertex or edge to keep track of all the vertices and edges the crawler C^{G_p} has visited. The delay variable $\tau_{C^{G_p}}$ is initially set to 0 and is too dynamically updated whenever C^{G_p} traverses an edge or an S-vertex (but not U-vertex–see below). When C^{G_p} is first initiated at v_m, it checks v_m's label $\ell_m = \{R^y_x, \ldots, R^l_k\}$ which contains information about the routes that v_m is part of and consequently finds its $\hat{\ell_m}$ next-hop neighbouring vertices, where

$$\hat{\ell_m} = \begin{cases} |\ell_m| & \text{if } v_m \notin \overleftarrow{\rho}^G, \\ |\ell_m| - k & \text{if } v_m \in \overleftarrow{\rho}^G. \end{cases}$$

and k is the number of times v_m appears in $\overleftarrow{\rho}^G$ (the number of routes in which v_m is a tail-end vertex; such routes terminate at v_m and thus do not have next-hop). However, since we are considering a multi-graph, it is possible that some of these routes are passing by the same next-hop neighbour (creating parallel edges between two vertices), so the $\hat{\ell_m}$ list may actually contain repeated vertices. If this is the case, we need

to consider each outgoing edge (even if all edges are parallel) separately because it may have different weight depending on which route it belongs to. Thus, once all next-hop neighbours are found, C^{G_p} selects one of them, say v_u, finds the edges (routes) between v_m and v_u, that is $\{e_i | e_i : v_m \rightarrow v_u\}$, and selects one e_i. Once an e_i is selected, C^{G_p} tags it as "visited", updates $\chi_{C^{G_p}}$ and traverses it. It is important that C^{G_p} tags any edge it traverses as "visited" so it does not revisit it again and enters in an infinite loop. Furthermore, if C^{G_p} arrives at a vertex v_k and found that there is only one unvisited edge e_i, it tags e_i as "visited", traverses it and then tags v_k as "exhausted" so it skips v_k if v_k ever happened to be a neighbour to some vertex C^{G_p} traverses in the future. Based on the type of the edges connecting v_m with its next-hop neighbouring vertices, C^{G_p} calls the appropriate mobility model (either PMDM, TTMDM or NTMDM) to calculate the delay of that edge, and updates its $\tau_{C^{G_p}}$ as follows $\tau_{C^{G_p}} = \tau_{C^{G_p}} + t_{v_x,v_y}$, where t_{v_x,v_y} is the delay returned for the edge $e_i : v_x \rightarrow v_y$ by the relevant mobility model (this applies to both R- and W-edges). Similarly, once C^{G_p} reaches an S-vertex v_y, it again updates $\tau_{C^{G_p}}$ but this time by calling MMM, such that $\tau_{C^{G_p}} = \tau_{C^{G_p}} + t_{v_y}$ where t_{v_y} is the delay assigned to v_y by MMM. However, since there is no transition between mobility models in U-vertices, MMM is not called when reaching a U-vertex. The crawler traverses the various routes by repeatedly *backing-up* whenever it finds a plausible or implausible route. The back-up procedure proceeds as follows: once a crawler finds an (im)plausible route, it checks its $\chi_{C^{G_p}}$ and traverses backward through the edge in $\chi_{C^{G_p}}[1]$ toward the vertex $\chi_{C^{G_p}}[2]$, where $\chi[n]$ is the n-th element of the list χ. It then deletes these two elements from $\chi_{C^{G_p}}$, and repeats the whole traversal process again (searching for neighbouring vertices etc.) but this time it does not traverse the edge it just came from because it is now tagged as "visited" (or generally any edge tagged as "visited"). The crawler C^{G_p} backs-up if: (1) $\tau_{C^{G_p}} + \epsilon > t_{v_n,v_m}$, or (2) traversed a vertex/edge that already exists in its $\chi_{C^{G_p}}$, or (3) v_n (the other end of the gap) is reached, where ϵ is a random value, or (4) it reaches a vertex v_j such that v_j is a tail-end vertex in all its routes (i.e. v_j is childless). In (1), the crawler baks-up once its $\tau_{C^{G_p}}$ reaches a value greater than t_{v_n,v_m} (the time difference between when the target was observed at v_m and later at v_n—the two ends of a gap G_p), and ϵ is a small constant. This means that the target would take much longer than t_{v_m,v_n} if it had traversed that route. In (2), we only accept loop-free routes because this is what a rationale target will opt to do (and also to prevent infinite loops), so if C^{G_p} reaches a vertex v_i such that $v_i \in \chi_{C^{G_p}}$, then it backs-up. In (3), once C^{G_p} reaches v_n, it checks its $\tau_{C^{G_p}}$, if $\tau_{C^{G_p}} + \epsilon \leq t_{v_m,v_n} - \epsilon$, it backs-up (in other words, if a crawler

returned a time much shorter than t_{v_m,v_n}, it is probably not the route the target has taken). Otherwise, if $t_{v_m,v_n} - \epsilon \leq \tau_{C^{G_p}} \leq t_{v_m,v_n} + \epsilon$, it backs-up, returns the route in $\chi_{C^{G_p}}$ as a possible route the target may have taken, as well as returning the corresponding $\tau_{C^{G_p}}$. Finally, in (4) C^{G_p} also backs-up when it reaches a *childless* vertex v_j; additionally, it tags v_j as "exhausted". The WBS algorithm terminates when its crawler terminates and that happens when the crawler reaches a vertex in which all neighbouring (next-hop) vertices are tagged as "exhausted", this means that they have been already extensively traversed (i.e. all their outgoing edges are tagged as "visited").

Proposition 2. *Given a finite search graph, the Weight-Bound-Search (WBS) algorithm will eventually terminate, with or without returning valid routes.*

Proof. Since the WBS is a weight-based algorithm, it is guaranteed to stop traversing a particular route R_i whenever its weight counter $\tau_{C^{G_p}}$ expires (i.e. $\tau_{C^{G_p}} \geq t_{v_m,v_n} + \epsilon$, where t_{v_m,v_n} is the delay through gap $G_p : v_m \rightarrow v_n$, and ϵ is a small constant). Thus the only way for the algorithm to run indefinitely is when it gets into an infinite loop and traverses the same route over and over again. However, a route R_i cannot be traversed more than once because the algorithm tags every visited edge and would not traverse any tagged edge, so as long as there is finite number of edges in a graph, the algorithm will terminate. □

In addition, the WBS algorithm will also terminate when traversing an infinitely *deep* graph because it traverses the graph down to the point when its weight counter $\tau_{C^{G_p}}$ expires. However, the WBS algorithm may fail to terminate when it runs over an infinitely *wide* graph (the node of the current level has infinitely many children) if none of the child of the current level has weight higher than $\tau_{C^{G_p}}$. This, nevertheless, contributes to the completeness of the WBS algorithm.

Proposition 3. *Given a finite search graph, the Weight-Bound-Search (WBS) algorithm is complete. If there exist one or more solutions, WBS will return them all.*

Proof. For a gap $G_p : v_m \rightarrow v_n$, a valid solution means that there is a route $R_i : v_m \rightarrow v_n$ with a weight τ such that $t_{v_m,v_n} - \epsilon \leq \tau \leq t_{v_m,v_n} + \epsilon$. The crawler C^{G_p} will traverse all valid and invalid routes and will terminates when there are no more edges to traverse. Therefore, if there is such solution route R_i, the crawler C^{G_p} will find it. □

Once the crawler terminates, and there are more than one route returned, the algorithm selects the *best-fit* route, such that $|\chi_{C_f^{G_p}}| =$

$min\{|\chi_{C_1^{G_p}}|, |\chi_{C_2^{G_p}}|, \ldots, |\chi_{C_n^{G_p}}|\}$. That is, the route with less hops will be selected because this is what a rationale target would probably do (choose a route that does not have many stops). Additionally, by observing the labels of the edges and vertices of the returned routes, a preferred route can be selected that minimises the number of transitions between different mobility models and/or curriers of the same model.

5. Conclusion

In this paper, we propose a multi-modal trace reconstruction algorithm that, given information about a partial list of a target's locations, it is able to reconstruct the full trace. For further evaluation of the algorithm, the reader is referred to the full version of the paper [1].

References

[1] S. Al-Kuwari, S. D. Wolthusen, Probabilistic Vehicular Trace Reconstruction Based on RF-Visiual Data Fusion, *Proceedings of the 8th IFIP WG 11.9 International Conference on Digital Forensics*, 2010. (full version)

[2] S. Al-Kuwari, S. D. Wolthusen, Probabilistic Vehicular Trace Reconstruction Based on RF-Visiual Data Fusion, *Proceedings of CMS '10*, vol. 6109, pp. 16–27, 2010.

[3] S. Al-Kuwari, S. D. Wolthusen, Fuzzy Trace Validation: Toward an Offline Forensic Tracking Framework, *Proceedings of SADFE '11*, 2010. (to appear)

[4] A. Dempster, N. Laird and D. Rubin, Maximum Likelihood from Incomplete Data via the EM Algorithm, Journal of the Royal Statistical Society, vol. 39, pp. 1–38, 1977.

[5] W.R. Gilks, S. Richardson and D. Spiegelhalter, Markov Chain Monte Carlo in Practice, Chapman & Hall/CRC Interdisciplinary Statistics, 1995.

[6] J. Harri, F. Filali and C. Bonnet, Mobility Models for Vehicular Ad Hoc Networks: a Survey and Taxonomy, IEEE Communications Surveys and Tutorials, vol. 11, pp. 19–41, 2009.

[7] D. Helbing and P. Molnar, Social Force Model for Pedestrian Dynamics, Physical Review E, vol. 51, pp. 4282–4286, 1995.

[8] M. Tanner and W. H. Wong, The Calculation of Posterior Distributions by Data Augmentation, Journal of the American Statistical Association, vol. 82, pp. 528–540, 1987.