Path Shapes – An Alternative Method for Map Matching and Fully Autonomous Self-Localization

Stefan Funke
FMI
Universität Stuttgart
Stuttgart, Germany
funke@fmi.uni-stuttgart.de

Sabine Storandt
FMI
Universität Stuttgart
Stuttgart, Germany
storandt@fmi.uni-stuttgart.de

ABSTRACT

We propose a novel scheme for map matching and fully autonomous self-localization. Our scheme is based on the unique characteristics of the shape of paths in a road network. As uniqueness of path shapes comes as no surprise in a world of infinite precision, we develop robust means of comparing shapes of paths under imprecisions. Even under this fuzzy comparison model, path shapes turn out to be sufficiently characteristic to allow for map matching or fully autonomous self-localization. We design an efficient data structure which allows for very fast path shape queries.

Categories and Subject Descriptors

E.1 [**DATA STRUCTURES**]: Graphs and Networks

General Terms

Algorithms, Data Structures, Localization

Keywords

Map-matching, Suffix Tree

1. INTRODUCTION

Nowadays, the majority of cars are equipped with navigation systems – sometimes via a device already built in by the car manufacturers, sometimes by dedicated navigation systems that are bought in the aftermarket or simply by the incorporated navigation system that comes with any Apple- or Android-class phone. In a typical application scenario the user expects the navigation system to guide her from the current position to some destination, giving turn directions when necessary. To achieve this, the navigation system needs to be aware of the current position of the car at any time. As of now, there are mainly three localization schemes which are most frequently employed:

GPS is a system of satellites that allow for a rather precise localization (up to few meters) using a GPS receiver.

ACM SIGSPATIAL GIS '11, November 1-4, 2011. Chicago, IL, USA

GSM when connected to a cell phone network, the current location can be estimated up to 50-500 meters from the positions of nearby cell phone base stations [7].

WLAN while acquiring StreetView data, Google also mapped and geocoded the IDs of Wifi networks along the way allowing for localization with a precision of 1-2 kilometers.

Given free sight to at least 4 satellites, a GPS receiver can determine a rather precise location in a few minutes. But free sight to GPS satellites is not always available, e.g. in urban environments or due to obstructing foliage. Furthermore, with GPS being administered by the US government, its precision can be deteriorated at any time (called *selected availability*). The other schemes require an active internet connection to interact with the servers of the network provider (GSM localization) or Google (WLAN or GSM localization) and do not provide sufficient precision for navigation purposes.

The procedure of pinpointing a possibly fuzzy location measurement to a concrete point on a map is called *map matching* and is of interest in other scenarios apart from self-localization. For example, the estimation of traffic flows can be accomplished on the network operator side by analyzing the imprecise GSM localizations of registered users and reconstructing their trajectories.

Main topic of this paper is the development of a novel localization scheme which is *fully autonomous*, i.e. does not require access to external information source like GPS, GSM or WLAN. Its precision is superior to any GSM or WLAN localization approach and comparable to GPS. We instrument the particular structure of the ambient space, i.e., the road network with the key idea being that we reduce the map matching problem to the problem of retrieving *path shapes* in a road network. A *path shape* is a description of a path as a sequence of relative movements, e.g.

50m straight, 45° left turn, 10m straight, 45° right turn, 100m straight, ...

Observe that neither an absolute direction nor absolute position information is contained in this description. The goal is to find in the road network a vertex starting at which this sequence of movements is possible (without leaving the road network). In Figure 1 we have depicted (partial) matchings for the path shape on the right. There are many small partial matchings but only one that is a full match. Several questions for this type of queries are of interest here:

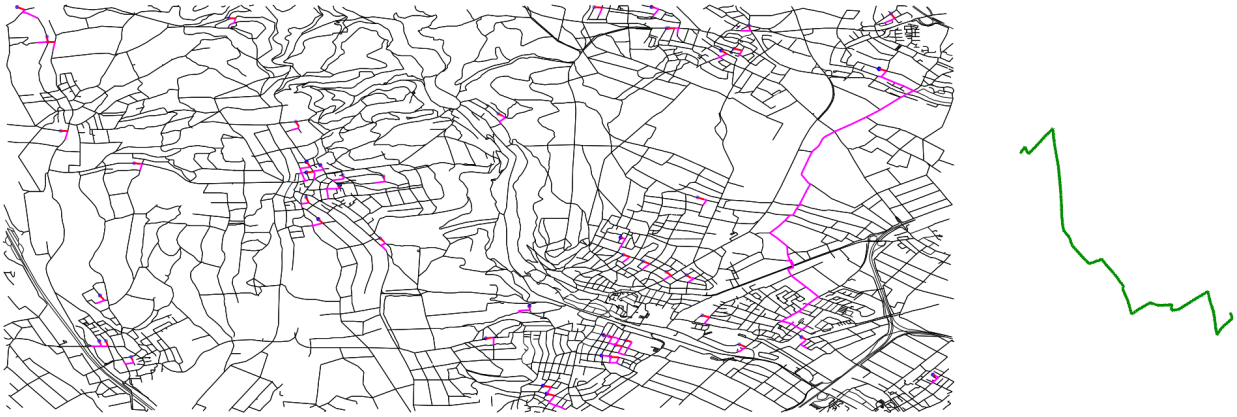

Figure 1: Path shape (green) and its matches (longer than 200m) in the Taunus map for a fuzzy equality model. Blue circles indicate path origins, first edges are marked red.

- how many such vertices exist for a given path shape? how long a path shape is required for uniqueness?
- how can a road network preprocessed such that path shapes can be efficiently retrieved?

Where can we expect to obtain such path shapes from? Most new cars are equipped with a complex onboard computer system which amongst others manages the *Electronic Stability Control (ESC)* System (*ESP* in Germany). ESC improves a vehicle's stability as it detects and minimizes skids by monitoring lateral acceleration, vehicle rotation (yaw), and individual road wheel speeds of the car. The very same data can be extracted via vendor specific protocols of the CAN-bus to obtain path shapes similar to the one above. Path shapes (of worse quality) can be extracted from the accelerometers of smartphones, which on the other hand might enrich the path shape description by absolute directional information (via built-in compasses) – we will come back to that towards the end of the paper.

1.1 Related Work

In all classical approaches to the *map matching* problem, we are given a sequence of (possibly imprecise) location measurements and aim at identifying the respective route in the road network and the current position. This problem is well-studied in different variations. The online version (measurements have to be processed the moment they are taken) is described amongst others by [9]. In the offline case the best possible path in the map for a given measurement sequence is chosen as the optimal one according to some scoring function. The score might for example be the Frechet-Distance, see [1], or the objective function value of an integer program [11]. In [4], the authors have shown that even very imprecise GSM localization allow for a very accurate reconstruction of the route a mobile user has travelled along in a network if measurements for a long enough period can be gathered. The respective algorithms are quite efficient, so a network provider could easily track millions of users in real-time.
Different from the problem dealt with in this work these papers are all concerned with input data containing *absolute* location information (though possibly very imprecise). Our problem is more some kind of *curve matching*, where for a given 'query curve' we are interested in identifying a similar curve in a collection of curves (e.g. implicitly given by an embedded graph) – *with translations and rotations being allowed.* Different approaches have been proposed for this problem, e.g. [6]. But most of them are very complicated and even if they allow for partial matchings (e.g. [3], [2]), they are not developed for comparing a reference curve with a very large set of curve shapes efficiently. Nevertheless ten years ago there was some effort in this direction [5] which has not found its way into real-world applications, though. This might be due to the limitations of sensorics at that time.

1.2 Our Contribution

With a path shape being the result of a physical *measurement* we cannot expect it to be precise and in particular to match exactly the path shapes present in the road network that we have at hand (typically in terms of an embedded graph). So we first have to come up with suitable path shape representations and means of imprecision-tolerant comparison of two path shapes before we can design an efficient preprocessing and retrieval scheme. We examine several road networks of North America and Germany with respect to their characteristics (under different representations and comparison methods) in terms of uniqueness of (shortest) path shapes. The results give rise to a novel scheme for preprocessing road network graphs such that efficient and robust (wrt to measurement errors) retrieval of path shapes and hence map matching or fully autonomous localization is possible. Our scheme is based in part on a reduction of our path shape retrieval problem to a *text retrieval* problem and employing generalized suffix trees (GST) [10].

2. MODELLING PATHS AS SHAPES

As a first step we have to think about how to model paths as shapes and what notion of "matching shapes/paths" is most sensible. Ground truth is always the path in an embedded graph $G(V, E)$ given as a polyline. Consider the ground truth path in Figure 2, left. It is very unlikely, though, that this ground truth path can be constructed exactly from (typically imprecise) measurements of any kind. The polylines that are reconstructed from measurements could for example look like the bold red polyline in Figure 2, center, or the bold blue line in Figure 2, right. *Typically the orientation of the blue and red polylines are not known, either, but are as-*

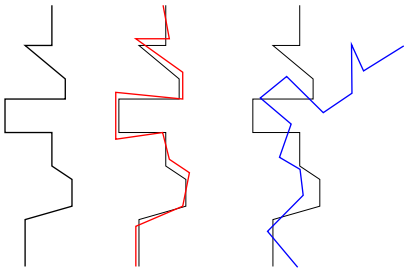

Figure 2: Global vs. Local Matching of Path Shapes: Ground truth (left), globally similar shape (center), locally similar shape (right). Important: typically the orientations of the red and blue paths are not known and are assumed to be known for visualization purposes only.

sumed to be known in this Figure for visualization purposes only. In the polyline in the center, the global shape of the polyline is still preserved whereas in the right polyline the global shape is destroyed even though locally the sequence of turns and straight line sections look very similar to ground truth. It depends on the type of measurement error which of the two notions of shape similarity are more appropriate. Our methodology developed in the following will deal with both. The notion of similarity as implied by the red polyline in the center of Figure 2 is pretty much what people have looked at in the area of *curve matching* according to Hausdorff or Frechet distance, for example. The notion of similarity implied by the blue polyline on the right is closer to what a verbal description of a path could look like: *Drive 100m straight, then make a sharp right turn, then another 100m straight, then turn left, followed by 50m straight. Then another turn to the left and 100m straight, ...* In a verbal description it is hardly possible to specify the turns precise enough that the general direction where one is heading is still correct. Still, this sequence of (sharp) turns and straight segments appears to bear enough information to allow for matching similar paths in a network. While this is not the main focus of the paper, the developed data structures could also be of interest to match such a verbal description of a path to a map.

In the following we will show how to model and compare polylines such that similarity – be it global or local – can be detected. It is obvious that for short paths the notions of global or local similarity approach each other. This will also be reflected in our experimental results later on.

2.1 Path Representations

Let us first determine a suitable representation of paths that will later allow for robust (with respect to measurement errors) comparisons. For a given path/polyline as a first step we *resample* the polyline uniformly by cutting it into pieces of let's say 1 meter each. A path/polyline of length l will hence be divided into l pieces. We choose two representations of this l-piece polyline $P = s_1, s_2, \ldots, s_l$, where s_i denotes its i-th segment:

localAngle-Representation (LAR): P is represented by the sequence of $l-1$ angles $(\angle l_{i-1} l_i)$ for $i = 2, \ldots, l$.

globalAngle-Representation (GAR): P is represented by the sequence of $l-1$ angles $(\angle l_1 l_i)$ for $i = 2, \ldots, l$.

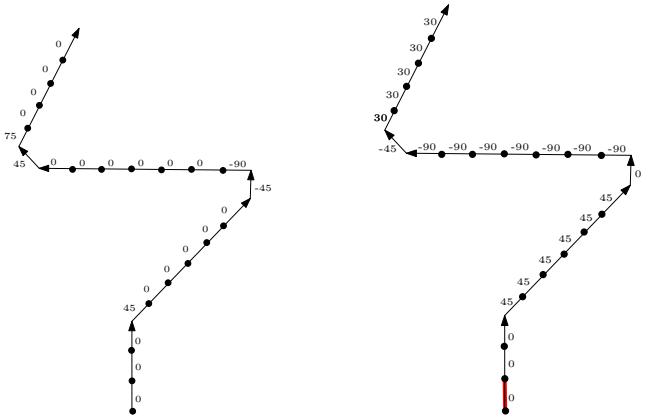

Figure 3: Example path. Left: LAR representation. Right: GAR representation relative to the first edge (red).

In both cases we round the angles to integral (degree) values. See Figure 3 for an example of the two representations of the same polyline.

The two representations differ from each other as soon as imprecisions of the measurements come into play. While distances can be measured quite accurately (one of the authors drives the > 800 km route from Stuttgart to Greifswald on a regular basis and experiences differences in the odometer readings of less than a few hundred meters when the same route is taken), measuring directional changes is more challenging. The second representation is suitable for measurements for which the directional error does not accumulate. For example, if distances are measured via an odometer, but relative changes of direction are always measured via the change of angle towards a very distant landmark, we will get a polyline in which the directional change is possibly imprecise but its error does not accumulate; also, the overall shape of the path is somewhat preserved. The first representation, on the other hand, is more suitable if directional changes might accumulate, e.g. when the change of direction is inferred from individual road wheel speeds (or in a different scenario, as a verbal description). The latter representation has the drawback that it might declare two path shapes similar even if they do not look similar at first sight on a global scale.

2.2 Robust Comparison of Path Representations

A naive way of comparing two paths π_1, π_2 represented as a sequence of angles $a_1, a_2, \ldots, a_l$ and $b_1, b_2, \ldots, b_l$ by GAR or LAR is to check whether $a_i = b_i$, $\forall i = 1 \ldots l$. Of course this will rarely lead to a match even if we have very precise measurement devices at hand.

A first approach to soften the equality condition is to use an **angle tolerance** $\mathbf{t}_\alpha$ and declare paths with representations $a_1, a_2, \ldots, a_l$ and $b_1, b_2, \ldots, b_l$ equal if $|a_i - b_i| \leq t_\alpha$. Still this condition seems too harsh, in particular, if e.g. due to different resampling, a sharp bend shows up at the i-th position in one representation of the same path whereas at position $i+1$ in the other representation, see Figure 4.

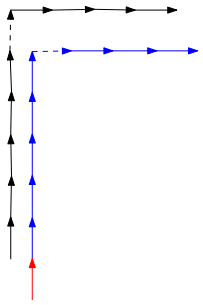

Figure 4: With the red edge the paths are equal. Without it the two dashed sections get compared and the paths are declared unequal unless $t_a \geq 90$.

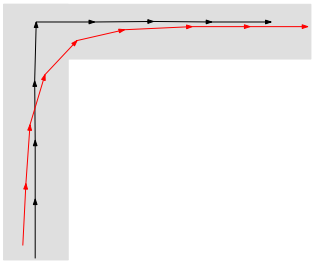

Figure 5: Black: Curve derived from the map. Red: The same curve reconstructed by measurements.

This problem of different resampling can be tackled by allowing a **'wobbling' comparison**, i.e. the paths are considered equal if there exists a function $\phi : \{1, \dots l\} \to \{1, \dots l\}$ with $\phi(i+1) \geq \phi(i)$ for $i = 1, \dots l-1$, $\phi(i) \in [\max(0, i - w), \min(i+w, l)]$ for some **range w** $\in \mathbb{N}$ and $|a_i - a_{\phi(i)}| \leq t_a$. The existence of such a function ϕ can be easily checked by a left-to-right sweep over the two path representations. Note that this also takes care of systematic under- or overestimation of distances in the measurements, so the function ϕ could also be thought of a regauging of the odometer.

Angle tolerances and wobbling do not get rid of the problem that a 90°turn might show up in one representation as a sequence of nine 10°turns whereas in the other as one single 90°turn, though. This is a quite natural problem which might not even be induced by measurement errors but by different ways drivers make a 90°turn, see Figure 5. Similarly, unexpected obstacles like potholes might incur artefacts in the path shape that prevent a proper matching. Depending on the representation – GAR or LAR – we will have to come up with different remedies.

In **GAR**, we allow small sections of the paths to differ. This can be achieved by defining a **range r** $\in \mathbb{N}$ and a **percentage c** $\in [\mathbf{0}, \mathbf{1}]$, such that for every section of the path p of length r at least $c \cdot r$ angles are equal to the other path's corresponding section according to one of the previous comparison approaches (exact *or* with tolerances *or* with wobbling). We call this approach **range-based comparison (RBC)**. Unfortunately, RBC only applies for **GAR**, since LAR is based on local deviations as main representation.

Instead of allowing $(1-c)r$ angles in each section of length r to disagree, in case of **LAR**, we introduce a moving average over the angles, so called **sum-based comparison (SBC)**. Two path representations A and B are declared equal if any path section of **length r** the sums of angle differences of A and B on that section are equal (*exact* or with **tolerance c**), see Figure 6. In case of LAR, SBC replaces the angle-by-angle comparison with tolerances but could be combined with *wobbling* to handle distance deviations.

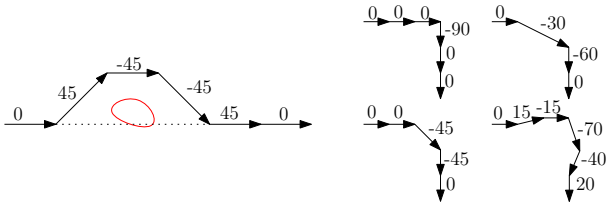

Figure 6: Left: Avoiding an obstacle (red). The sum of the angle differences is 0 in conformity with the direct path (dotted). Right: Four possibilities for driving a curve, all resulting in a sum of -90.

3. CHARACTERISTIC PATH SHAPES ON REAL-WORLD GRAPHS

Our goal is to use (imprecise) path shapes for map matching and self-localization in a road network. For this goal to be achievable at all, we first need to make sure that paths in real-world networks are sufficiently 'characteristic', that is, paths that are long enough have a unique path shape – even under imprecision-tolerant comparisons.

To that end we performed the following experiment: First we pick a random shortest path $\pi = v_1, v_2, \dots, v_k$, construct its angle representation $p = a_1, a_2, \dots, a_l$ (LAR or GAR). Now we are interested whether there exists another shortest path $\pi' = w_1, w_2, \dots, w_{k'}$ with angle representation $p' = b_1, b_2, \dots, b_{l'}$ which shares a long, 'similar' (according to the comparisons defined in the previous section) prefix. Or, in other words we are interested in finding the minimal i such that the prefix $a_1, a_2, \dots, a_i$ is unique amongst the angle representations of all shortest paths in the network (if such an i exists). The following modified Dijkstra when called on each vertex in the graph – we call it 'shape-preserving Dijkstra' (SPD) – achieves exactly this. SPD starting at a vertex v proceeds like a 'normal' Dijkstra, but pushes a vertex w into the priority queue only if the angle representation (LAR or GAR) of the path from v to w is equal (according to our comparison function) to a prefix of $a_1, \dots, a_i$ of $a_1, \dots, a_l$. The call of SPD on v essentially only explores paths as long as they have a similar LAR/GAR prefix as π. The length of the unique prefix then corresponds to the length of the longest explored path when SPD has been started on every vertex in the network (excluding the path π, of course).

Note, that using fuzzy comparisons we might not be able to always find such a prefix if there are several possible sources, see Figure 7. Hence in that case we let our procedure return the minimal prefix that contains a unique suffix.
We used three test graphs for evaluation: The German Taunus ('T', small graph), the road network of Massachusetts ('MA', graph of medium size) – particularly interesting because it contains many of grid-like subgraphs and whole Germany ('GER') as an example of a large graph. The main figures of these graphs can be found in Table 1. Identifying the unique prefix for every possible (maximal) shortest path is too time-consuming, even for the German Taunus. Therefore we restricted ourselves to a large number of (s, t)-pairs chosen uniformly at random, for which we first computed the shortest path using normal Dijkstra. Then we started for every node a SPD computation and let them run until all PQs are empty.

For a choice of comparison models the average prefix lengths as well as maximum prefix lengths and timings can be found in Table 2 sub-divided according to the used path model. Moreover the plot in Figure 8 shows the average prefix length for a wide range of comparison parameters for *GAR*. Figure 9 shows for Massachusetts, that prefix lenghts are actually concentrated around their mean. In Figure 10 one can get an impression how the number of matching paths in relation to a reference path decreases, if the number of considered edges grows. It looks like an exponential decrease for all used comparison models, which explains the shortness of unique path prefixes. Interestingly the graph of Germany (more than 50 times larger than the one of Massachusetts) exhibits shorter unique prefixes, probably because of the less 'planned' structure of the road network due to historical reasons.

	nodes	edges	avg path length
T	11220	24119	5.6 km
MA	294345	731874	101.5 km
GER	18575544	38798358	452.5 km

Table 1: Number of nodes and edges of our benchmark graphs, along with the average path length in these graphs in kilometers.

	GAR	avg query time (sec)	avg prefix length (m)	max prefix length (m)
$t = 0$	T	0.02	72	604
	MA	1.13	246	2869
	GER	86.24	169	1607
$t = 5$	T	0.03	125	1280
	MA	1.68	575	8439
	GER	94.75	274	4148
$t = 10$	T	0.05	355	4781
$r = 50m$	MA	2.36	3084	32337
$c = 0.9$	GER	134.37	684	8430

	LAR	avg query time (sec)	avg prefix length (m)	max prefix length (m)
$t = 0$	T	0.03	74	599
	MA	1.31	279	4480
	GER	90.24	182	1351
$t = 5$	T	0.03	123	1561
	MA	1.85	641	23497
	GER	95.35	287	5047
$t = 10$	T	0.11	222	1840
$r = 50m$	MA	3.73	1413	41036
	GER	144.45	946	78796

Table 2: Unique prefix lengths in different sized road networks, with t denoting the angle tolerance and r (and c) being the parameters for RBC or SBC. Query time (in seconds) denotes the time until prefix uniqueness has been established by SPD for an $s - t$ path. Query time and the prefix length (in meters) are average values of 1000 random queries.

In any case, even for rather relaxed similarity measures, path shapes become unique quite early, e.g. for LAR, angle tolerance $t_\alpha = 10^\circ$ over a range of 50 meters, the average length of the characteristic prefix of a shortest path is only 946 meters. So path shapes in principle seem to be a viable means for localization in a road network. Note that our SPD algorithm is already an algorithm to perform this localization. The running times for larger networks are prohibitive, though (note that starting a SPD at every single vertex of the network scales badly with the network size). In the next section we will present a data structure which allows this localization several orders of magnitudes faster.

One might wonder whether examining only *shortest paths* is too severe a restriction. In case of map matching e.g. for traffic estimation this is less of an issue since users tend to move on at least piecewise shortest paths, see also [4], but also for the application scenario of a navigation system this seems ok; even when wandering around one tends to do so on at least piecewise shortest paths. If those pieces are at least as long as the characteristic prefix length, restricting to shortest is fine. So in the following we will focus on the preprocessing and analysis of *shortest paths*, even though in principle we could also consider all possible paths – this will be the focus of future work. Furthermore we see from Table 2 that using LAR or GAR does not really make that much of difference – which comes as no surprise since for short paths, LAR and GAR are essentially equivalent. We will focus on GAR in the following (mostly due to historical reasons), exactly the same techniques apply for LAR, the results are very similar, too.

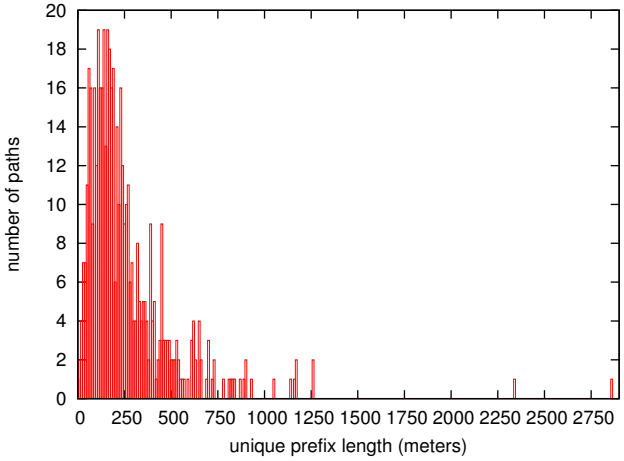

Figure 9: Distribution of unique prefix lengths for MA with $t_a = 0$.

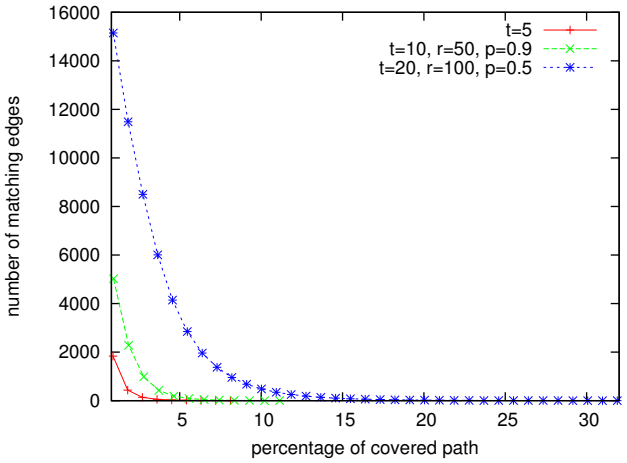

Figure 10: Number of matching edges of the Taunus graph in relation to the position on the reference path for some selected models.

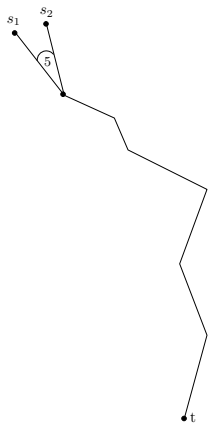

Figure 7: Source not identifiable for a comparison model with $t_a >= 5$.

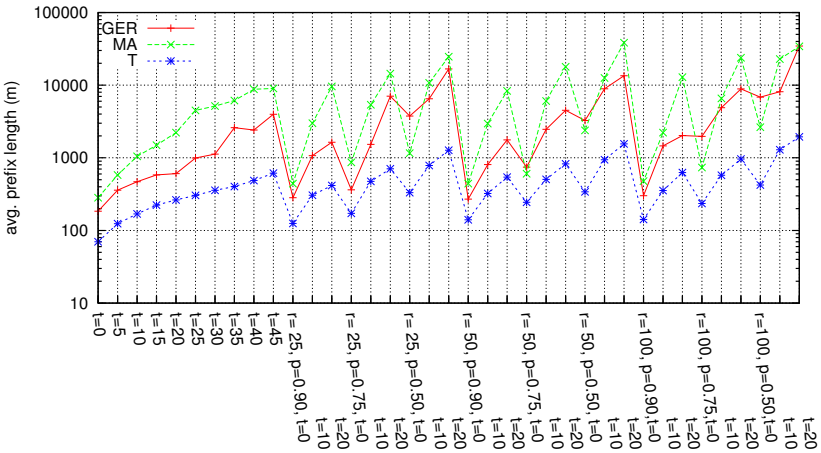

Figure 8: Average prefix lengths under different comparison models for 1000 examples each.

4. EFFICIENT PREFIX HANDLING USING GENERALIZED SUFFIX TREES

If we consider the sequence of angles defining a path as a concatenated string, classical string search algorithms can be used to identify a path in the 'text' which describes the whole road network.
To find a string pattern of size m in time $O(m)$ in a text of length n, $n \gg m$, the latter has to be preprocessed. One method to do so is constructing a (generalized) suffix tree of the text. A GST represents all suffixes of a set of strings $S_1, \cdots S_k$, fulfilling the following characteristics:

- each tree edge is labelled with a non-empty string
- there is no inner node with degree 1
- any suffix of a string corresponds to a unique path in the tree (starting from the root) with the concatenated edge labels along that path starting with this suffix

A suffix can be grafted into a GST by first identifying its longest prefix that already exists in the tree by tree traversal. If the path of this prefix ends in a leaf node, we just add the remaining characters of the suffix to the according edge label. If the path ends in an inner node, we have to create a new edge and a new leaf representing the last part of the suffix. Also the path could end implicitly, that means the according edge label contains additional characters, that are not in the suffix. Hence this edge has to be split and so its label, such that we derive a new inner node that represents the longest prefix. Then we can proceed as described before. By performing this for every suffix occurring in $S_1, \cdots, S_k$ we obtain a GST of this set of strings. Note that there are more efficient ways of constructing GSTs, see [10], but as we will have to determine the set of strings $S_1, \ldots, S_k$ online, this more efficient construction does not apply for us.

With the GST being constructed, the question whether a given pattern is contained in $S_1, \ldots, S_k$ can be answered in time linear in size of this pattern, if the alphabet size is bounded, as we can associate an array with every node, which for each letter of the alphabet stores the edge (if any) whose label starts with this letter. So a query starts in the root of the GST, looking for an outgoing edge with a label, that begins with the first 'letter' of the pattern. If such an edge exists, we have to compare the respective edge label to the related pattern prefix element by element. If they are equal, we can go to the end point of the edge and repeat the search with the remaining elements of the pattern. If we always find a match, the pattern is contained in the underlying set of strings. The number of its occurrences equals then the number of leaves of the subtree beneath the last visited node. If at some point no match can be found, the pattern is not contained in $S_1, \ldots, S_k$.

For a practical implementation, some tricks can be applied. For example, long straight parts of a path induce long sequences of the same 'angles' (both LAR or GAR), which can be stored in a compressed manner without affecting construction and properties of the GST. While not improving any theoretical worst-case bound, the practical performance is greatly improved.

4.1 Online GST Creation

Our required GST has to contain every minimal unique path prefix occurring in $G(V, E)$. But going through all possible paths and identifying in each case the unique prefix with the method described in Section 3 is far too time consuming.

OBSERVATION 1. *A prefix of length l is unique if it is not equal to any other occurring prefix with length $\leq l$ for the employed comparison model.*

Using this simple observation we can develop an efficient approach to create the GST. We start a Dijkstra computation from every $v \in V$, stopping only when the first element in the priority queue (PQ) has a distance label greater than a given maximal distance d_{max} (or the PQ became empty). Then this Dijkstra run is frozen, i.e. we keep the actual priority queue, as well as the distance and predecessor labels. Now for all vertices that were pushed into the PQ we compute the LAR/GAR for the respective path up to a maximal length of d_{max} starting from the source vertex. Note, that for each vertex w with $dist[w] > d_{max}$ we do not have a real

endpoint of the prefix in the map and therefore have to use an implicit vertex instead.
The representations then get grafted into our tree T. To that end, every node has a tag, whether its unique, and a pair of vertex IDs, marking the start and endpoint of the respective path (-1, if the endpoint is implicit). If a node is revisited with a different end point, its initial true unique tag is set to false (implicit nodes are always false tagged). If the endpoint is the same but the start point differs, we have the situation as depicted in Figure 7. Here the node stays unique but we add the additional source to the node. Having performed this for all frozen Dijkstra computations, we check in T for the newly created nodes, if they are labelled unique. For every unique node the search subtree below the respective target vertex does not need to be considered anymore for any related source, pruning the search space of the respective Dijkstra run considerably. As long as there are non-empty PQs of some Dijkstra runs, we repeat this procedure, while increasing d_{max} in every round (see Figure 11). Note, that there might be superfluous nodes and edges in T after each round, namely the subtrees beneath nodes labelled unique as well as edges and nodes referring to implicit nodes. These nodes and edges can be deleted. At this

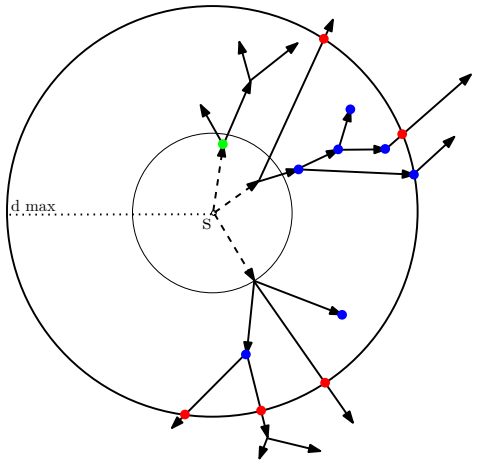

Figure 11: Prefix elongation from source s. The smaller circle indicates the previous maximal distance. Blue points mark nodes, which were settled in the current round. Red points show implicit nodes, that have to be considered to assure prefix uniqueness on 'real' nodes. The green point marks the endpoint of an already unique prefix, therefore the respective subtree can be ignored.

point we have created a GST on the necessary prefixes of path shapes, which is sufficient to answer *exact* path shape queries. As queries will be based on inherently imprecise measurements, we have to think about how our GST data structure is of use for similarity searches.

4.2 GST – Fuzzy Tree Traversals and Construction

So far we have learned how to construct a GST under a imprecision-intolerant comparison model. Of course, our goal is to construct and use our GST in a imprecision-tolerant manner. To that end let us first describe a *fuzzy tree traversal*, whose goal it is to determine all strings in the GST which are similar (according to one of our imprecision-tolerant comparison notions) to some given query string $a_1, \ldots, a_l$. We obtain those strings by traversing the GST from the root, exploring not only the single path which matches but all paths which are 'similar' to the query string. Unfortunately, creating a GST as described in the previous section together with fuzzy tree traversal does not really solve our problem, as a typical query will then return *several* strings. Our GST creation procedure only made sure that under *exact* comparisons the path prefixes were long enough to guarantee uniqueness, for imprecision-tolerant queries, longer prefixes might be necessary.

To that end we have to modify the procedure for deciding whether a node in the GST is unique or needs to be further explored. In case of exact queries we get this information by a simple tree search, for fuzzy queries we have to employ the above described fuzzy tree traversal each time we add a path and mark all nodes inside the matching tree as non-unique if the respective target does not equal the one on the new path. This would increase construction time considerably, so we apply a more efficient approach. Additionally we attach to each node in the GST the length of the respective prefix as well as a pointer to its parent node. This allows for checking for a certain path in the tree if it is similar to any other contained path by selecting all nodes with an attached length greater or equal to the length of this path and performing a backwards tree traversal as long as the two paths are similar according to the employed comparison model. Hence we can decide *at the end of each round* for every newly created node with a unique tag whether this tag is feasible. This improves running time, as the number of unique nodes after grafting all path codes up to a certain length into the tree is considerably smaller than the number of temporary unique nodes during the prefix elongations.

4.3 Correctness and Runtime

Correctness. Our algorithm for building the GST is fundamentally different from common GST constructions like [10] in that the set of strings $S_1, \ldots, S_k$ we want to represent in the tree is not given explicitly right from the start but is identified during construction. Therefore it actually requires proof, that our output tree is a GST.

LEMMA 1. *The created tree T is a GST.*

PROOF. Assume $p \in T$, but $q \notin T$ with q being a suffix of p. $q \notin T$ implies that q has a unique prefix which is strictly shorter. But then p has to have a strictly shorter unique prefix as well and therefore $p \notin T$, contradiction to our assumption.■

According to Lemma 1 we can employ the standard machinery for (exact) string search in GSTs. As the alphabet in our case size $O(1)$ (there are only 360 integral angle values) we obtain a query time of $O(l)$ for a query of length l.

Runtime. The algorithm consists of three phases:

I. Identifying shortest paths in the graph via Dijkstra computations

II. Graft all shortest path encodings into the GST

III. Extract all unique nodes from the GST und update the according PQs, clean the GST from superfluous nodes and edges

If all Dijkstra computations run until the end, we need $O(n^2 \log n + nm)$ for phase I. Grafting a path representation of length l into the GST takes time $O(l)$, overall we might have $\Omega(n^2)$ paths of length $\Omega(n)$ and therefore a total runtime of $\theta(n^3)$ for part II. In the last phase there can be only $O(n^2)$ paths, which equals the total effort for that part. So overall we can only guarantee a runtime of $O(n^3)$, but this is not the running time experienced in practice, since there the unique prefixes tend to be very small (see previous section). So far, we have not striven for a better theoretical running time.

4.4 Answering Queries

At query time, we are given a candidate path shape as some polyline ; we transform this path shape to the desired representation (LAR or GAR) and then employ the constructed GST to identify a source vertex s for this path shape in our map by performing a (fuzzy) tree traversal until identifying a unique node. As we are not only interested in the starting point and the end point of the unique prefix, but its corresponding path in the map, we can run a single SPD from the identified starting point. In contrast to the naive employment of n SPD computations this reduces the running time by orders of magnitudes.

In large networks, even this single SPD execution could be too expensive, as its running time is typically superlinear in the length of the longest path explored. In this case, a natural speed-up technique is the piece-wise reconstruction (similar to the one employed in [4] for map matching): If a path is very long, we expect the unique prefix to be relatively short. That means, if we consider the remaining path without this prefix, it still should be long enough to contain a unique prefix on its own. Again we can find this prefix by using the GST. Therefore we can reconstruct the path piecewise by identifying the paths of the unique prefixes one after the other and concatenate them. If there remains a non-unique tail, we have to start a SPD in the last correctly identified vertex to reconstruct the whole path.

5. EXPERIMENTAL RESULTS

In this section we first discuss how the map preprocessing can be performed efficiently in practice. Afterwards the query times for exact and fuzzy queries are evaluated. Finally we introduce quality metrics for the map matching problem and analyze how accurate our approach is in relation to the density and the precision of the given measurements. Our implementation is written in C++, timings were taken on a single core of an AMD Opteron 6172 with 2.1 GHz and 96 GB RAM. In the following tables we restrict to the GAR model, since – as seen in Section 3 – for short paths they are essentially equivalent. We have conducted the same experiments with the LAR model and experienced very similar results.

5.1 Preprocessing

We build generalized suffix trees for all three test graphs. In order to do so, we need to choose an initial value for d_{max} and decide how it should be increased in every round. The larger the growth rate, the more paths have to be considered in one round. Because we cannot decide which of them already contain unique prefixes before the end of the round, there can be many superfluous encodings, tree traversals, creation of new nodes and edges as well as deletions. Choosing the growth rate of d_{max} very small, we might have rounds where no new paths are feasible at all, still we have to spend time on that. Figure 12 shows the preprocessing time for MA depending on an *additive* growth rate for d_{max} (added after each round). The minimum is near $150m$, so we chose this as additive growth rate. The preprocessing time can be even improved, when employing this constant growth rate until 75% of the Dijkstra runs have completed and double the growth rate in each round afterwards.

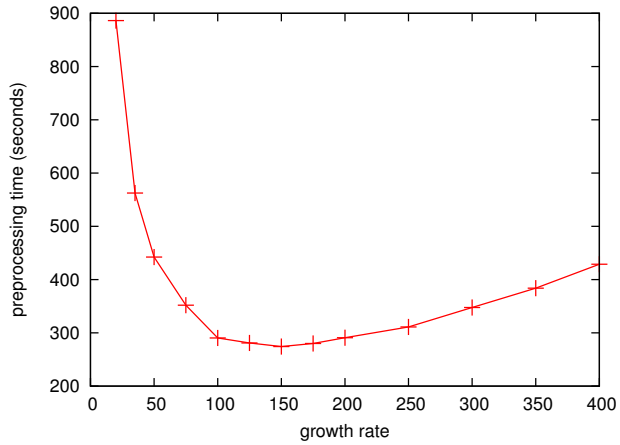

Figure 12: Preprocessing time for the MA graph depending on of the growth rate of d_{max}.

In Figure 13 we see that the number of completed Dijkstra computations (PQ ran empty) depending on the search radius follows a typical saturation curve. Up to $1250m$ there is an exponential increase, so 85% of the Dijkstra runs have completed at this point; the remaining 15% runs have finished when the search radius exceeds $3750m$. The resulting

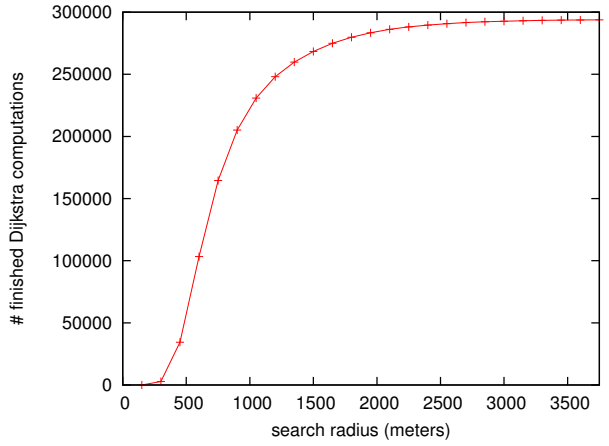

Figure 13: Number of completed Dijkstra computations versus search radius for the online GST creation of MA.

preprocessing time for all three benchmark graphs can be

found in Table 3 along with the final size of the according GST and the maximal search radius, that was necessary to make all prefixes unique. The effort required for preprocessing might look prohibitive at first sight (for Germany $2-3$ days on a single core and using about 70 GB of RAM on our server), but remember that this preprocessing time has to be spent only *once*. The resulting data structure is less than 2 GB in size, hence comparable to the road network representation itself, and can easily be stored even on mobile devices with microSD cards (typical size 8 GB). As the parallel Dijkstra computations are inherently parallelizable, we expect further speed-up in that respect.

		T	MA	GER
$t_a = 0$	time (sec)	2	185	13105
	# nodes (GST)	51793	2593809	86271669
	search radius (m)	1350	5293	4680
$t_a = 5$	time (sec)	11	1701	114487
	# nodes (GST)	77784	2789902	117638920
	search radius (m)	2703	16350	9125
$t_a = 10$	time (sec)	46	3781	196580
$r = 50$	# nodes (GST)	135265	3805956	697329542
$c = 0.9$	search radius (m)	5004	42800	12750

Table 3: Experimental results of the online GST creation for exact and fuzzy paths.

5.2 Query Times

We presented two approaches to answer a query, i.e. to identify a given path shape in the map: In Section 3 we described the naive way to tackle the problem, by starting a SPD in each vertex and waiting until all these runs have finished. Using GSTs to extract the path's source as described in Section 4, we can answer a query using only a single SPD, starting in that particular source. The query times of these approaches are collected in Table 4 for some selected comparison models. Timings contain path encoding as well as the summed runtime of all necessary SPDs and where appropriate the time for performing a (fuzzy) tree traversal. Unsurprisingly the naive approach ends up

		T	MA	GER
$t_a = 0$	naive	0.0235	1.1355	86.241
	GST+SPD	0.0002	0.0008	0.027
$t_a = 5$	naive	0.0347	1.6810	94.750
	GST+SPD	0.0010	0.0720	0.304
$t_a = 10$	naive	0.0522	2.3649	134.374
$r = 50m$	GST+SPD	0.0035	0.1318	2.233
$c = 0.9$				

Table 4: Overview of query times for selected comparison models using different map matching approaches. Timings (in seconds) are averaged over 1000 random queries.

last for all considered inputs. Employing the GST for identifying the source and subsequent path exploration using a single SPD leads to a speed up of factor of 3000 for answering exact queries in the Germany graph.

5.3 Accuracy

To evaluate the accuracy of our approach we performed two different experiments: On one hand we checked under which comparison models an exact path from the map can still be identified uniquely. On the other hand we simulated imprecise measurements by perturbing and subsampling the input data and then asked for which comparison models the correct path in the map still matches this input.
To measure the quality we compare our resulting path p' to the correct path p using the same metrics as in [8]:

- $A_N(p, p')$ denotes the percentage of edges of p that are *not* matched by p'
- $A_L(p, p')$ denotes the percentage of the length of p that was *not* covered by p'

The only possible reason for an exact path not to be identified correctly is the path being too short and hence occurring multiple in the map. This means we either get $A_N = A_L = 0$ in case of a unique match or $A_N = A_L = 1$ otherwise. For fuzzy queries our quality measures can take any value in $[0, 1]$, because apart from paths being too short, source and target vertices of a path can be ambiguous under the chosen comparison model. For the evaluation we chose the road

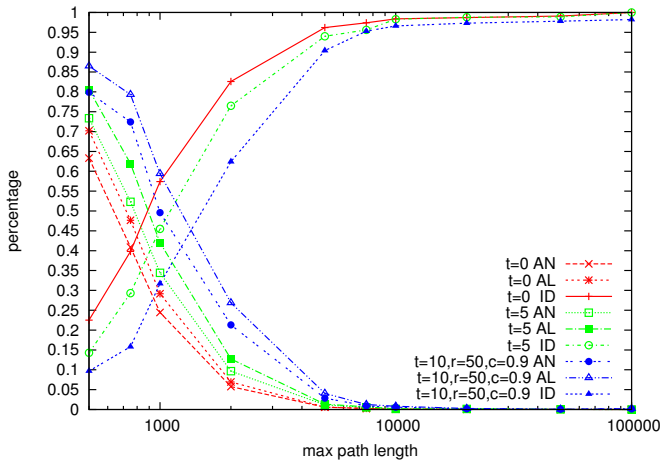

Figure 14: Quality of our approach measured for some selected comparison models. *AN, AL, ID* are averaged over 1000 random queries with AN=A_N, AL=A_L and ID denoting the percentage of exactly identified paths.

network of Massachusetts, because retrieving path shapes is most challenging here due to the grid-like substructures. In Figure 14 one can see the quality of our results for paths derived from the map, that are restricted by their lengths. As to be expected, the error rate is high for short paths but improves rapidly with growing path lengths, identifying almost 100% of the paths totally correct even for fuzzy models.

Normally the input won't be a polyline describing exactly a path in the map, but the angles might differ due to measurement uncertainty, the length of the driven route might vary slightly and depending on the density of the measurements we might miss some turn changes. If the angle and length deviation is bounded by a constant, we can use these values directly as angle tolerance and allowed wobbling range and therefore we can still guarantee to identify the path, if it is long enough. The results of the simulation of such queries can be found in Table 5. At first we randomly added to each angle a value in $\{-5, -4, \cdots, 4, 5\}$ (row 1:angles ± 5). The resulting path shape is typically not present in the map,

hence without an angle tolerance we are not able to identify the path. Next, we perturbed the edge lengths randomly, but restricted the new total path length to differ not more than a half percent from the original one (average 1 km deviation for MA). Only a comparison model using wobbling (here with a range of 250 m) enables us to match the path in the map. Finally we simulated different measurement densities. Sparse measurements lead to smoother polylines, that exhibit quite big differences to the original path on small sections. Therefore only a model with range-based comparison leads to usable results.

Perturbation	$t_a = 0$	$t_a = 5$	$t_a = 10, r = 50$ $c = 0.9, w = 250$
angles ±5	0.989/0.991	0.000/0.000	0.000/0.000
length ±0.5%	0.881/0.931	0.864/0.888	0.005/0.001
density 1 m	0.000/0.000	0.000/0.000	0.001/0.000
density 2 m	0.966/0.973	0.942/0.940	0.002/0.003
density 5 m	0.991/0.995	0.977/0.982	0.014/0.017
density 10 m	0.993/0.998	0.992/0.995	0.058/0.061

Table 5: A_N/A_L for several input perturbations and comparison models (averaged over 1000 examples).

6. FUSION WITH OTHER INFORMATION

If additional information apart from the relative movement pattern is available, the problem gets easier, of course; for example, many mobile devices nowadays have an **electronic compass** built-in which can be used to enrich the path shape information. In this case, we would naturally encode the path as sequence of (absolute) directions; the number of paths with the same shape representation shrinks drastically compared to the models considered so far, see Figure 15. Unsurprisingly this results in much faster preprocessing and query times.

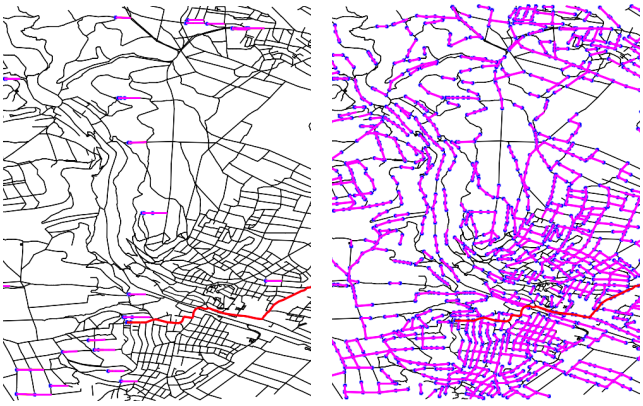

Figure 15: Reference path (red) and its matches under the comparison model $t_a = 5$, left with compass information, right without (GAR).

Our approach can also be combined with imprecise but **absolute location information** in a natural way by excluding possible source/target vertices in the GST traversal while answering queries. Car manufacturers have been using techniques to incorporate relative movement patterns for their self-localization, they are only meant as a backup for a precise localization method like GPS during short periods of time (e.g. while driving through a tunnel), though. We can also make use of very imprecise location information like 'we are currently in the state of Bavaria'.

Furthermore, **height information** can also be acquired fully autonomously (at least about the *change* of height via a barometer) and incorporated into a description of a path shape. While probably not of much use in the midwest of the US, the height profiles of routes in the Alpes might be characteristic even without any (horizontal) directional information.

7. CONCLUSIONS

We have presented a novel approach for self-localization and map matching which is based on the acquisition and retrieval of relative movement patters which we call *path shapes*. While directional information so far has been used only as a backup for short periods of time when e.g. GPS is unavailable, we have shown that it is fully sufficient for self-localization and map matching on its own. Natural future directions of research are reduction of the preprocessing times, consideration of *all* possible paths instead of shortest only, and incorporation of other information sources.

8. REFERENCES

[1] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *J. Algorithms*, 49:262–283, 2003.

[2] E. M. Arkin, L. Chew, D. Huttenlocher, K. Kedem, and J.S.B. Mitchell. An efficiently computable metric for comparing polygonal shapes. In *1st Symp. on Discr. Algorithms(SODA)*, pages 129–137, 1990.

[3] M. de Berg and A. Cook IV. Go with the flow: The direction-based frechet distance of polygonal curves. In *Proc. 1st Int. ICST Conf. on Theory and Practice of Algorithms in Computer Systems (TAPAS)*, 2011.

[4] J. Eisner, S. Funke, A. Herbst, A. Spillner, and S. Storandt. Algorithms for matching and predicting trajectories. In *Proc. of the 13th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2011.

[5] ERTICO. Agora website. http://www.ertico.com/agora-website, 2000.

[6] M. Frenkel and R. Basri. Curve matching using the fast marching method. *EMMCVPR*, pages 35–51, 2003.

[7] Mohamed Ibrahim and Moustafa Youssef. Cellsense: A probabilistic rssi-based gsm positioning system. *CoRR*, abs/1004.3178, 2010.

[8] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. Map-matching for low-sampling-rate gps trajectories. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 352–361, 2009.

[9] M. Quddus, W. Ochieng, and R. Noland. Current map-matching algorithms for transport applications: state-of-the art and future research directions. *Transportation Research Part C: Emerging Technologies*, 15:312 – 328, 2007.

[10] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995. 10.1007/BF01206331.

[11] H. Yanagisawa. An offline map matching via integer programming. In *Proc. 20th International Conference on Pattern Recognition (ICPR)*, pages 4206–4209. IEEE, 2010.