Eur. Transp. Res. Rev. (2017) 9:38
DOI 10.1007/s12544-017-0254-8

ORIGINAL PAPER

An approach to geometry-based dynamic location referencing

Rüdiger Ebendt[1] · **Louis Calvin Touko Tcheumadjeu**[1]

Received: 27 July 2016 / Accepted: 20 June 2017

Abstract
Introduction An important requirement for knowledge infrastructures in smart cities is the continuous updating of location-based information. Protocols for dynamic location referencing like e.g. OpenLR or AGORA-C tackle the problem of accurately matching locations between dissimilar digital maps. They are map-agnostic and aim at limiting the amount of descriptive data to reduce bandwidth. However, there are applications for which the weaker requirement of map-independence is completely adequate, and for some there are even no restrictions in bandwidth (e.g. in the EC-funded project ROSATTE, and in the DLR projects MobiLind and KeepMoving), and with relaxed constraints it is possible to learn from methods in similar areas like road network matching and map conflation, in order to achieve a more accurate solution. Following this path, this paper presents a map-independent approach developed in the ongoing DLR project I.MoVe, which can be combined with a bandwidth-efficient dynamic location referencing method like e.g. OpenLR to target applications with bandwidth restrictions.
Methods The proposed new approach works line-oriented and is guided by a measure of geometric dissimilarity. It is a top-down approach, recursively splitting up the source route into parts, thereby following a divide-and-conquer strategy to reduce the problem until it can be solved trivially.
Results It is currently capable of mapping closed line locations (i.e. circular routes, representing either the boundaries of areas or the tours of e.g. delivery trucks) from a TeleAtlas map to a NAVTEQ map on-the-fly with a success rate of 97.5% (OpenLR: only 82.5%), and also capable of mapping short line locations (i.e. linear routes) on-the-fly between the same maps, with a success rate of 99.7% (OpenLR: 91.9%).
Conclusion In conclusion, the new approach to match linear or circular routes between two dissimilar maps is highly accurate and map-independent, but access to both involved maps is required. The approach can also be combined with a bandwidth-efficient dynamic location referencing method like e.g. OpenLR to obtain a compact format before the descriptive data is transmitted.

Keywords Location based services · Digital road maps · Dynamic location referencing · OpenLR · Road network matching

1 Introduction

Municipal leaders, politicians, civic planners and other key stakeholders in cities are now looking to enable the vision of a “knowledge infrastructure”, i.a. by deploying location based services. Location based services can help in better governance, planning and functioning of smart cities. An important requirement is the continuous updating of location-based information for transport users.

With many questions arising during this mission, there is a strong interest in protocols for dynamic location referencing

---

This article is part of Topical Collection on Smart cities and transport infrastructures

✉ Rüdiger Ebendt
Ruediger.Ebendt@dlr.de

Louis Calvin Touko Tcheumadjeu
Louis.ToukoTcheumadjeu@dlr.de

[1] German Aerospace Center, Institute of Transportation Systems, Rutherfordstr. 2, 12489 Berlin, Germany

like e.g. OpenLR and AGORA-C: they are addressing the problem of accurately matching locations between dissimilar digital maps. They are needed since “conventional geo-referencing methods using coordinates or a pre-defined set of identifiers have structural limitations; they may fail to match the same location in different maps due to discrepancies with respect to level of detail, spatial and temporal accuracy, and semantic dissimilarities” [18]. For example, location references based on road names and house numbers could fail due to misspellings and notation differences, see e.g. [27].

Location references solely based on geographical coordinates (e.g., WGS84 coordinates) also have severe problems. Often there are significant offsets between different maps or topological differences due to varying accuracy of the digitalisation from analogue map data, different manufacturing methods, and changes of the road network over time [27]. Therefore, a geographical coordinate located on a road in one map might be situated off-road in another. Moreover, there may be roads which only exist in one of two different maps (this holds even for different releases of maps from the same vendor).

For example, an intersection, modelled as a simple intersection in one map, might be represented by a complex intersection in another map. An additional problem is raised from differences in the representation of roundabouts: since some vendors use patterns to represent them, a roundabout might be exactly circular in a first map, whereas in a second one the roundabout was digitalised in its real world geometry which can be elliptically [27]. Finally, irrespective of the aforementioned deviations between maps, a location reference solely based on a geographical coordinate may not be unique. For instance, take the case of referencing locations on two motorways crossing without allowing access to each other. At the place of the crossing, the same coordinate would stand for two separate locations on different motorways [27]. In all these cases a geographical coordinate is not sufficient for a reliable location reference.

Moreover, there are also problems with the well-established RDS-TMC service: location codes are stored in a predefined TMC location table. This results in the need to incorporate this common set of location codes in all map databases. Therefore the TMC approach requires a significant amount of overhead and maintenance efforts over time, see e.g. [18, 27]. For these structural and practical reasons, the use of RDS-TMC is restricted to European major roads, such as the motorway network, the national main roads, and only some important parts of urban roads. This means that large portions of the road network are left uncovered, whereas every location in a map (including locations covering minor roads) can be transferred using a *dynamic* location referencing protocol like OpenLR or AGORA-C: such protocols describe how to match locations between dissimilar digital maps on-the-fly [18, 27]. They are *map-agnostic* in the sense that

(a) they are map-independent, i.e. they do not require maps from a specific vendor
(b) neither the encoder nor the decoder knows about *both* used maps, but rather the encoder only accesses the source map, and the decoder only accesses the target map.

There are many use cases for dynamic location referencing. For example, the transferred information can consist of the current traffic situation at a certain location, a traffic forecast or special alerts [32]. In a typical use case for a smart city, such location content must be shared effectively between applications for navigation, transportation management, or wireless location-based services.

On the other hand, all approaches to dynamic location referencing have the disadvantage of possibly failing to properly encode and decode a location in some cases. Also, since no pre-coded location tables are used, a dynamic referencing method needs to transmit more information per location than a conventional method. Therefore, a dynamic location referencing method needs to achieve an acceptable level of performance in terms of

(1) hit rate
(2) data size per location reference (message size)

Hit rate is the percentage of “hits” during dynamic location referencing, where a “hit” means either a correctly identified location or a location correctly identified as not being present in the decoder map [43]. As a generally agreed industry goal, a dynamic location referencing method should perform at a hit rate of 95% and message size should be below 50 bytes on average [17, 27, 43].

The AGORA-C method is reported to have achieved the industry goal of an acceptably small message size and a sufficiently high hit rate [42, 43].

TomTom has provided test results for OpenLR, e.g. using a source map from TeleAtlas and a target map from NAVTEQ: success rates of 93% were achieved both for TMC paths and non-TMC paths [29]. Here, “success rate” is the percentage obtained by only counting the correctly decoded locations, i.e. those where encoder and decoder location were equal, and dividing this number by the number of all locations that could be decoded (for a discussion of the difference to the definition of “hit rate”, see Section 4.1).

In contrast to other approaches to dynamic location referencing like AGORA-C which have licensing fees meaning extra cost, e.g. [37], OpenLR is a royalty-free, open standard under a creative commons license (CC BY-ND 3.0 [5]), and therefore fosters interoperability and promotes free choice between different vendors and technology solutions [12].

Summarised, current methods for dynamic location methods either imply the extra cost of licensing fees, or do not fully meet the aforementioned industry goal yet, or suffer from both these drawbacks. The starting point for the present paper then is the observation that for a number of applications a map-independent (cf. part (1) of the previous definition of "map-agnostic") dynamic location referencing method is completely adequate. That is, an omniscient matching centre with access to both maps can be used. Moreover, even part (1) of the aforementioned industry goal, i.e. a message size below 50 bytes on average, is not relevant, if bandwidth is not restricted: e.g. in the aforementioned EC-funded project ROSATTE [25], static data is sent from a public authority to a map provider, and in two DLR projects, MobiLind and the ongoing project I.MoVe, route-based data from a traffic information provider is transformed from one well-known map to another, and often this transformation takes place on the same server. For such applications, it is possible to learn from methods in similar areas like road network matching (RNM) and map conflation, in order to achieve a more accurate solution.

A first approach is to obtain a permanent static mapping by use of RNM: since an omniscient matching center has access to both maps, a RNM algorithm (e.g. [23, 44]) can be used to establish such a mapping from the source map to the target map. It provides $M : N$ mappings ($M \geq 1$, $N \geq 0$) of every road segment in the source map, represented by a graph edge, to zero or more road segments in the target map. This mapping can be used to map a queried route in the source map to a corresponding route in the target map on the fly by "glueing together" the individual matches for the edges of the route.

Throughout the paper, this approach will be referred to as the "RNM-based approach". This approach is appealing, but has several drawbacks: establishing a static mapping by RNM means

(i) using a less accurate algorithm not tailored towards the mapping of routes
(ii) high maintenance cost because of the necessary preprocessing, and longer time for a transition to a new pair of maps;
(iii) extra memory requirement for the static mapping, higher initialization time since for a good performance the mapping must be reloaded into main memory every time the server is restarted;
(iv) high implementation effort, due to the lack of available open source implementations.

Regarding point (i) it can be noted that e.g. the most recent algorithm DSO in [44] aims to match fragmented linear objects and matching areas. However, the matching certainty and accuracy is hard to be ensured in this case as the fragmented objects provide little contextual information [44]. The main use case in several projects of German Aerospace Center is dynamic location referencing (DLR), i.e. traffic information like e.g. the information about congested stretches of road is location-referenced using short routes. In such a DLR-context, a successful match always means a complete match, i.e. partial matches are not of interest and are considered as a failed match (see also the introduction to Section 4). Moreover, a qualitative comparison in Section 4.2.4 shows that the algorithm proposed in this paper has a higher success rate than DSO, due to the fact that it considers the route as a whole rather than just glueing together individual edge matches.

Regarding point (ii), notice that e.g. [23] reports a run time of around two hours to match 100,000 road segments on a standard IPC. The time for an additional interactive editing of the results may even take several days.

This paper presents an approach called "Geometry InterMapMatching Extension" (GIMME) which follows this path but aims to overcome the aforementioned disadvantages. It has been developed in the ongoing DLR project I.MoVe. It assumes that the requirement of a map-agnostic method (in the sense of the previous definition) can be dropped, but still provides a map-independent approach. It is tailored towards the on-the-fly matching of short routes, with respect to both execution speed and accuracy. GIMME is a dynamic method, i.e. it does not use a static mapping. Therefore no preprocessing is required. Consequently, setting up or resetting a matching center for a new pair of maps can be done much faster, and requires less memory. GIMME can be expected to have a better success rate than the RNM-based approach (see Section 4.2). On the other hand, the error detection rate of GIMME can be lower than for the RNM-based approach. However, the first quality measure, i.e. the success rate, is a much more important measure in practice (see also Section 4.2.4). If the error detection rate is of concern, GIMME can also serve as a temporary replacement for the RNM-based approach during its preprocessing step when resetting to new maps.

Neither GIMME nor the RNM-based approach is targeting minimal bandwidth. Nonetheless, both approaches can be combined with a bandwidth-efficient dynamic location referencing method like e.g. OpenLR to address applications with bandwidth restrictions (see Section 3.5). GIMME is currently capable of mapping closed line locations (i.e. circular routes, representing either the boundaries of areas or the tours of e.g. delivery trucks) from a TeleAtlas map to a NAVTEQ map on-the-fly with a success rate of 97.5% (OpenLR: only 82.5%), and also capable of mapping short line locations (i.e. linear routes) on-the-fly between the same maps, with a success rate of 99.7% (OpenLR: 91.9%). When mapping longer linear routes consisting of 20 road segments between the two maps, GIMME showed a success rate of 98.0%, whereas the estimated success rate of the RNM-based approach with DSO (“$\mathrm{RNM_{DSO}}$”) is only 92.3%, a difference of 5.7%. For routes with 25 segments, GIMME has a success rate of 93.8%, whereas the estimated success rate for $\mathrm{RNM_{DSO}}$ is only 90.5%.

The paper is structured as follows: in Section 2, necessary background from location referencing, road network matching, map conflation, and a geometry-based approach to matching of single edges between maps is given. The latter approach is extended to a new algorithm matching complete routes in Section 3. Next, in Section 4, the accuracy of the new algorithm is compared to that of OpenLR and of $\mathrm{RNM_{DSO}}$. Finally, the work is concluded in Section 5.

2 Related work and background

To keep the paper self-contained, necessary background and related work is briefly reviewed in this section.

2.1 Types and use cases of locations

Typically, the following locations are distinguished [18, 32]:

- point locations,
- line (or linear) locations, and
- area locations.

Point locations are zero-dimensional elements in a map that specify a geometric location. An example for a service involving point locations is the search for nearby point-of-interest (POI) objects. Other examples are e.g. locating petrol stations and speed cameras [32].

Line (or linear) locations are one-dimensional elements in a map describing roads or a list of connected roads, and are used during e.g. route guidance: recommended routes are calculated based on the current traffic situation. That way, route guidance is adjusted to incoming information such as traffic jams. In this, dynamic traffic information is incorporated into route calculation, and the respective additional new information is location-related (e.g., information about a congested stretch of road). Every such location-based information must be transferred to some processing unit, often a server at a centre, performing the route calculation. In turn, the calculated recommended routes then serve to redirect traffic flows in order to relieve overloaded roads, and clearly they are location-based information themselves. Recommended routes are displayed on websites or in mobile applications, and often a reliable way to transfer them from a sender, e.g. the centre, to the respective receivers, e.g. the PNDs, is also needed for them.

In the OpenLR standard, area locations are two-dimensional parts of the surface of the earth which are bounded by a closed curve. An area location may cover parts of the road network but does not necessarily need to. Examples for area locations not covering the network are areas describing woodland, a sea or an agrarian country. Other more typical use cases for area locations are: a Wi-Fi hotspot with its signal range, weather information, weather reports about e.g. average rainfall for every cell of a grid, low emission zones, areas affected by weather or environmental conditions (bad weather, smog), flood areas, areas that are congested (due to any cause, e.g. traffic overload, public event, or disaster), administrative areas, pedestrian areas, large crowds of people, areas that are blocked for traffic, and areas subject to city toll [32].

According to the OpenLR White Paper v1.5 [32], “OpenLR can handle locations which are bound to the road network but also locations which can be everywhere on earth (not bound to the road network)”. If they are not bound to the road network, a location is represented by sequences of WGS84 coordinates (sometimes plus additional parameters describing the extent of the location). In OpenLR, the only area locations that are bound to the road network are closed line locations. They reference the area defined by a closed path (i.e. a circuit) in the road network [32]. The boundary always consists of road segments. Besides areas, a closed line location may also describe the course of a tour, e.g. a tour of freight vehicles starting and ending at a depot, or even the course of a marathon.

In AGORA-C, area locations are represented either in terms of concatenated line locations representing the area’s outline (i.e., explicitly composed) or by means of a set of locations of any type contained within the represented area (i.e., implicitly composed) [18].

2.2 Dynamic location referencing

This section gives a brief overview of previous work on dynamic location referencing. Related work includes the ILOC concept (“Intersection LOCation” [9]), developed in the EVIDENCE project (“Extensive Validation of IDENtification Concepts in Europe” [24], 1997–1998), extended ILOC [2], SPOT (“SPacial Object Tags” [7]), ROSA (“Reconstruction of Objects on a Second mAp” [6]), the Routing Point or Pivot Point approach of Siemens VDO [16], the Extended Geometry or GOODLANE approach of Bosch [15], the AGORA location referencing method, developed in the AGORA project (“Implementation of Global lOcation Referencing Approach” [8, 19], 2000–2002), with its successor AGORA-C [18, 42, 43] developed in the mobile.info project ([4], 2004–2007), MEI-LIN (“MEthod for Identifying Locations In road Networks” [41]), and OpenLR (hinting at “Open Location Referencing”)[32].

An ILOC (Intersection LOCation) is a junction where two or more intersecting roads with different road descriptors (e.g., different street names) meet. In the ILOC approach, the location of the ILOC is directly referenced by a WGS84 coordinate, and then the direct reference is extended with additional information like three road descriptors of connecting roads. This helps to avoid ambiguity [9]. The ILOC concept was implemented and tested in the EVIDENCE project, and hit rates of about 80% were achieved [24, 43].

The extended ILOC approach uses the ILOC codes of the location's bounding intersections for determining the expansion of the location [27], and at the same time is based on the geometry of a location and descriptive elements [43]. Of note is that a variant of extended ILOC has been developed into TPEG-Loc, part 6 of the TPEG Generation 1 standard, which had been issued as ISO standard ISO/TS 18234-6 [8, 20, 22, 33].

ROSA, developed around the time of the AGORA project, is in essence a variant of Extended ILOC, and maximum hit rates of about 85% were reported [6, 43]. According to [43], "[...] the Routing Point approach [is based] on connectivity and route calculation, and the Extended Geometry approach on the matching of characteristic geometry". According to [27], "the main idea of these approaches [i.e., extended ILOC, Pivot Point, and GOODLANE] was to select special points of the road network and supply them with certain attributes to make them easily identifiable". For a detailed description of these methods, see [8].

The SPOT approach of Bosch generates three components of a location reference: position, object type, and descriptor. After transmission of the reference, a "search window" is generated around the position in the decoder map. Then, all objects of the requested object type are requested within the search window, the given descriptor is verified with the descriptors of the collected objects, and finally that object in the search window is selected which matches the descriptor [7].

In the AGORA project an attempt was made to combine three dynamic location reference methods, namely the aforementioned extension of the ILOC approach, the Pivot Point approach and the GOODLANE approach. The resulting hit rate was good enough to meet the stated industry goal in terms of a 95% hit rate [17, 27], but the message size was unacceptably high [42, 43]. Therefore in the successor AGORA-C (the "C" stands for "compact", akin to a reduced message size, and also hints at "ALERT-C", the official name for the TMC standard [43]), complexity was reduced by focussing on two approaches only, the ILOC extension and the Pivot Point approach. The AGORA-C method is reported to have achieved the industry goal of an acceptably small message size and a sufficiently high hit rate [42, 43]. It is of note that the method is in the process of being accepted as ISO standard ISO/DIS 17572-3 [21], which was successfully approved as a Draft International Standard by ISO and will be issued as an International Standard pending final approval [18, 37].

Rather than aiming at reduction of message size, MEILIN puts more emphasis on a better identification of the surrounding road network of the location in order to facilitate identifying the correct lanes. As the key idea, "[...] one special point is chosen which is either in or close to the location to be referenced. Rooted at this point, a tree is built up in the surrounding road network following certain rules. The vertices of the tree are points on connected roads which have a certain driving distance from the root" [27]. After transmission, the coordinate is placed into the decoder map. Within a certain search area around that coordinate potential root points are chosen on the map, and "the same algorithm as on the encoder side is used to build up a tree rooted at each of the chosen points" [27]. After that, the resulting trees are matched with the transmitted information. Since the tree can grow large if a longer location needs to be encoded, splitting the locations and using multiple smaller trees might be necessary [27].

OpenLR is an open standard for dynamic location referencing, i.e. for encoding, transmitting, and decoding location references in digital maps [31]. It was launched by TomTom International B.V. in September 2009 and developed for the use case of transferring traffic information from a centre to in-vehicle systems, built-in or used as an add-on (PND, Smart Phone). OpenLR is published as an open-source framework, including a reference implementation of the proposed standard (in contrast, no reference implementation is publicly available for any of the aforementioned approaches). Everyone is invited to contribute to this open source software project and to enhance the existing solution.

The standard addresses the problem of accurately matching locations between dissimilar digital maps, enabling a reliable data exchange for many different types of location information, and cross-referencing through maps of different vendors, versions, or different cartographic standards [31]. OpenLR has been certified by the Travellers Information Services Association (TISA) [35] as an industry standard (TPEG-OLR [34]) which is part of the UML-based TPEG structure generally known as TPEG Generation 2 (TPEG2). Moreover, the OpenLR extension for DATEX II [11] makes it possible to use DATEX II [10] with an extended location referencing model containing OpenLR. OpenLR has a growing number of users [30]. Users include the Swedish Transport Administration who is using OpenLR in its DATEX II services, GEWI who develops TIC, a commercial off-the-shelf software platform used to process data for information services, TrafficNet who is South Africa's largest supplier of Traffic and Travel News to the Broadcast and Media Industry, and other leading companies in the field of vehicle tracking, traffic-related data warehouses, and geographic information systems (GIS). OpenLR has also been used in project ROSATTE (“ROad Safety ATTributes exchange infrastructure in Europe” [25], 2008–2010): a first benefit was that no licensing fees had to be paid, and a second, that the effort needed to implement encoders and decoders was significantly reduced because a framework of Java components is provided [28]. The results were positive: for example, OpenLR location references created from two different maps describing a spatial extend of a speed limit (update), could be decoded on a third independent map with average location matching rates of over 90% [28, 36].

The main idea of OpenLR for locations which are bound to the road network (such as line locations or closed line locations, describing an area bounded by a closed path, i.e. a circuit of road segments) is covering the location completely with a concatenation of (several) shortest-paths. Each shortest-path is specified by information about its start line and its end line, given as so-called location reference points (LRPs): an LRP describes an object in a digital map which consists of coordinates and additional information about a line in the map, and every single LRP specifies a position or a line in a digital map. The resulting set of LRPs uniquely identifies the location. The path described by such a location reference (the “location reference path”) may be longer than the original location and offsets trim this path down to the size of the location path [32].

2.3 Road network matching

This section briefly outlines background from the three most-cited works in road network matching. The reviewed methods are usually used in the context of combining two distinct maps into one new map (map conflation). Moreover, a previous geometry-based approach to match single edges between two maps, “Geometry Matching”, is briefly reviewed. The results of this method are the starting point for the proposed approach GIMME. The first and the third method, i.e. “Iterative Closest Points” and the “Delimited-Strokes-Oriented Approach”, have mainly been included to complement and complete the understanding of the second method, “Buffer Growing”. Buffer growing, a line-oriented bottom up method, is the approach most similar to GIMME, which is a recursive top-down method (see Section 3).

2.3.1 Iterative closest point

Iterative Closest Point (ICP) was formulated in [1] as a generic algorithm to match two clouds of points by minimizing the difference between them.

Subsequently, it has been adopted in the field of road network matching [38]: first all nodes of the two networks are extracted and then correspondences of nodes between the two resulting datasets are established by combining initial thresholds for dissimilarity measures based on the Euclidean distance, the number of incident edges (i.e. the valence), and the angle differences between emanating edges. Next, if line segments are enclosed between two nodes of one dataset, they are matched to those enclosed by the corresponding nodes. The rationale is that if two roads from different datasets have corresponding start and end nodes, then there is a high likelihood that the roads themselves are corresponding counterparts [44]. The algorithm re-computes the initial thresholds based on data of the first computed node correspondences, and then iterates the approach with stepwise relaxed constraints. This works well when matching whole roads since this only requires establishing the node correspondences between intersections. However, finding the respective correspondences for the remaining start and end nodes in two digital road maps from different vendors is more difficult. The reason is that the geometry of road segments (e.g., length and the number of segments per road) varies strongly among different maps. In contrast to the line-oriented BG algorithm in Section 2.3.2, ICP is designed for the larger data sets involved when matching large portions of two road maps and therefore seems to be more appropriate for map conflation than for matching individual line or area locations.

2.3.2 Buffer growing

Buffer Growing (BG) [39] is an efficient algorithm for the general task of line matching [44]. BG accounts for the aforementioned problem of possible significant differences between corresponding line segments with respect to their start and end points and their lengths (cf. Section 2.3.1) by introducing the concept of a growing buffer. At start, $1{:}N$, $N \geq 1$ correspondences between line segments are considered: a spatial buffer around the one source segment is defined, and for a valid correspondence all N target segments must be completely confined to this buffer. If no fitting target segment can be found in the current buffer, it is stepwise expanded until a respective number of target segments can be found [40]. This process also facilitates computing $M{:}1$ or $M{:}N$, $M > 1$ correspondences (i.e., matching of complete routes): for this purpose, new logical integrities are built from previously found correspondences, which are then subject to new applications of BG in subsequent steps.

After the Buffer Growing process, a list of potential candidates for the matching reference is computed. The list may be ambiguous and typically contains a large number of matching candidates. By computing the geometric and topologic similarity between each matched pair, the best matching candidate can be confirmed as the final solution [45].

2.3.3 Delimited-stroke-oriented approach

The basic idea of the Delimited-Stroke-Oriented (DSO) approach is to exploit contextual information as much as possible. For this purpose, a kind of pre-processing step identifies potential "[...] *fundamental elements* at more abstracted levels", i.e. a "series of conjoint road objects [chained together and acting] as the fundamental element in the matching process" [44]. For this purpose, the Delimited Strokes are progressively constructed at three different levels: on the first level, a Delimited Stroke represents a series of connected segments which have "good continuity" to each other (which means that one segment follows the other in almost the same direction), and are delimited by "efficient terminating nodes" (which are either prominent crossings with at least 4 incident nodes or dead-ends), whereas on the second level it is an arbitrary series (i.e., sharp turns are now allowed) which is delimited by arbitrary crossings or dead-ends. Only on the third level single road segments (edges) are considered.

Experimental results showed that the outlined contextual approach increases the accuracy of road network matching significantly [44]. Of course, this algorithm is designed for matching a whole map to another (i.e., for map conflation), and requires a significant amount of pre-processing. It is left to show that there can also be benefits for the on-the-fly matching of single line or area locations between dissimilar maps (i.e., for the more focused problem addressed in this paper).

2.3.4 Geometry matching

In [26], digital road networks have been benchmarked with respect to their fitness for route finding and traffic simulation. For this purpose, a geometry-based approach was used to match single edges (not complete routes) in a source map to one or more edges in a target map (thereby establishing $1{:}N$ but not $M{:}N$, $M > 1$ relationships). In the following, the basic ideas of this algorithm called Geometry Matching (GM) are given briefly.

A geometry-based dissimilarity measure for pairs of source and candidate edges is given which is mainly based on three criteria:

(i) average distance
(ii) angular difference, and
(iii) length of the mutual projection of the candidate and the source edge,

respectively. At first, a proximity search is done in the target map within a certain radius around the position of the start node of the source edge, and the dissimilarity measure is computed for the found edges. All edges with a dissimilarity measure above a certain threshold are excluded from the candidate list. To determine a $1{:}N$, $N > 1$ relationship for a source edge, the first best edge is determined as the candidate with the highest measure, and the result list is initialised as the singleton with this best edge. Then, a forwards and a subsequent backwards search are performed in an iterative manner, using the topology of the target map:

Firstly, the candidate list is re-initialised with those outgoing edges of the current best edge, for which the dissimilarity measure is good (i.e. low) enough, and ordered with respect to the measure. Then, a new best edge is calculated and appended to the result list, and so on until an empty candidate list is encountered. Secondly, an analogous process starts with the incoming edges of the current best edge until termination. In the end, the result list is either empty if no candidates could be found, or is a sequence of one or more connected edges forming a stretch of road in the target map corresponding to the source edge.

Summarised, the GM method resembles the idea of the BG algorithm (see Section 2.3.2), although no building of new logical integrities from previously found correspondences takes place here, and consequently no $M{:}N$, $M > 1$ correspondences, i.e. no matches for complete routes are constructed. It is worth mentioning that GM uses both geometric and topological attributes of the edges. The use of a threshold for the average distance between source and candidate edges is an interesting novelty (see Section 4.1 for the details), since previous methods were using measures on top of the Hausdorff or the Fréchet distance.

3 Geometry InterMapMatching Extension (GIMME)

This section introduces the proposed new approach called "Geometry InterMapMatching Extension" (GIMME). It is divided into five subsections: Section 3.1 presents the algorithm, and Section 3.2 concludes the presentation of the algorithm with further remarks. GIMME uses GM to calculate the list of potential candidates, and therefore, the presentation of GIMME is concluded by a brief review of GM's thresholds for exclusion of a candidate in Section 3.3. Then, Section 3.4 demonstrates the presented algorithm by a worked example. Next, Section 3.5 discusses how GIMME can be combined with a bandwidth-efficient dynamic location referencing method like e.g. OpenLR to target applications with bandwidth restrictions.

3.1 Algorithm

In this section, first the idea of the algorithm is derived from key observations made for the problem at hand. Next, pseudo-code is given for the algorithm GIMME. Regarding the variable names in the pseudo code, capital Latin letters are used for sets (e.g. for a set of solutions, i.e. a set of edge lists), whereas small Latin letters are used for set elements, associative arrays, lists, and for natural numbers.

Similar to the BG algorithm (see Section 2.3.2), it works line-oriented and is guided by a measure of geometric dissimilarity. Another analogy is that (partial) solutions are only considered when they are topologically connected. However, in contrast to the BG algorithm, which is expanding and combining partial solutions bottom up, GIMME is a top-down approach, recursively splitting up the source route into parts, thereby following a divide-and-conquer strategy to reduce the problem until it can be solved trivially.

Divide-and-conquer algorithms are known to make efficient use of memory caches: once a sub-problem is small enough, it and all its sub-problems can be solved within the cache, without accessing the slower main memory [14].

GIMME builds upon the results of the GM algorithm (see Section 2.3.4): GM computes an ordered list of candidate edges (or "candidate list" for short) for every edge of the source route (or "source edge" for short), and returns an associative array relating every source edge to a corresponding candidate list. The order in each such candidate list is with respect to the dissimilarity to the source edge, see Section 3.3. The aforementioned array is essentially a "pool" of eligible candidate edges, and the objective of GIMME is to construct a best match for the whole route by drawing (zero, one, or more than one) candidate edge(s) from the pool for every source edge, and by finding the best possible combination of them. The following key observations lay the ground for the new algorithm:

(i) Firstly, in the context of this paper, source and target routes must be directed paths, i.e. finite sequences of edges connecting a sequence of vertices where the first and the last vertex may coincide. It is of note that cycles have not been excluded here, and that the edges must represent road segments which are connected in such a way, that a vehicle can drive on them in the driving direction, that is, from the start of the first segment to the end of the last segment.

(ii) Secondly, there must be a non-empty candidate list for every source edge: if this is not the case, then there is at least one source edge which cannot be matched to any edge in the target map. Consequently, also the whole route cannot be matched. Notably, partial matches are not of interest here, complying with the fact that the intended use case is dynamic location referencing (see the introduction to Section 4).

(iii) There are $M{:}N$ relations between source and target edges. This means that for a single source edge $1{:}n$ relations must be established, where $N \geq n \geq 0$. For an example of the case $n = 0$, consider e.g. a 3:2 relation between source and target edges: if all source edges are mapped to at least one target edge, then there must be one destination edge occurring twice in the resulting target route. This means that this target route cannot be homologous. Notice that the fact that a source edge might not become mapped to any target edge does not contradict (ii): also in this case eligible candidates for the respective edge do exist, but all these candidates are "consumed" for other (i.e. incident) source edges.

(iv) Of note is that the order of a candidate list has nothing to do with that of the edge sequence of a directed path. This is because the candidates are ordered with respect to dissimilarity to the source edge. Therefore, GIMME considers the *set* of candidates in a candidate list, i.e. the order of the elements in the list is not used (cf. Algorithm 2, line 9). To build the final target route by combining partial sequences drawn from the pool of candidates for every individual source edge, GIMME considers all permutations of subsets of a candidate list, including the empty set because of the case $n = 0$ in (iii). Since this seems to introduce a very complex step to the algorithm, the following observation is exploited to ensure efficiency of the approach:

(v) Since the final route must be a directed path, it is not necessary to consider partial sequences for which it is already known that this property does not hold.

(vi) Point (v) gives rise to the following recursive formulation:

1. The final target route is constructed by trying to connect partial routes (i.e., partial solutions) for the head and tail of the list of source edges. Whenever a current partial solution for the head can be connected topologically to a partial solution for the tail (cf. Algorithm 2, line 32), then we have a match for the whole source route, and this solution is stored as a potential solution.
2. More precisely, the partial solutions for the head are computed by subsequently considering all permutations of subsets of its candidate list (cf. Algorithm 2, line 13 and line 18). To cover the case $n = 0$ in (iii), GIMME also considers the empty subset as a possible (partial) solution for the head, but only if the current head solution can also be used as the prefix of a solution for the tail. A test for this is performed in Algorithm 2, line 28. If it succeeds, GIMME stores the corresponding solution for the tail (i.e. without a preceding match for the head) as a possible solution (cf. Algorithm. 2, line 29).
3. Only connected permutations (i.e. directed paths) are considered (cf. (v) and Algorithm

   2, line 14). The partial solutions for the tail are computed by invoking the outlined algorithm recursively on the tail. Every recursive call returns a list of all (partial) solutions found (where a solution respects (v) by construction), and the calling code then checks each returned solution for the tail for topological connectedness with the current solution for the head (cf. Algorithm 2, line 32). The boundary case of the recursion is reached when the tail has shrunk to an empty list (for which an empty list of solutions is returned, cf. Algorithm 2, line 3).

4. Finally, from the list of solutions for the whole route returned at the topmost calling level, the best solution must be chosen. Currently, the best solution is defined as the largest admissible solution (i.e. one that is a longest list) for line locations (cf. Algorithm 1, line 11), and as the first found cyclic path for closed line locations (cf. Algorithm 1, line 13). A largest admissible solution is preferable because such a solution is more likely to cover the whole source route (i.e. to not miss any part of it). For an explanation of the term admissible, see (vii). In the (rare) case that there is more than one largest admissible solution, currently one is simply picked randomly.

(vii) A solution is admissible, if and only if (a) it respects (ii), i.e. for every source edge at least one corresponding candidate edge is used in it (but not necessarily for that respective source edge, see the case $n = 0$ in (iii)), and (b) the length of the solution does neither exceed 120% nor fall below 80% of the length of the source route matched by this respective solution.

Notice that a solution is already a directed path by construction; hence this is not required again in (vii). This also holds for partial solutions. That way partial solutions are excluded early if they cannot be completed to a solution for the whole route, and are never considered again at a shallower calling level of the recursion.

This helps keeping the list of partial solutions returned by a recursive call (and all efforts depending on the size of this list, corresponding to the effort of a divide-and-conquer algorithm for combining the partial solutions) small. To reduce the number of recursive calls, the result of each recursive call for a list of source edges e is cached, using e as the key (cf. Algorithm 2, line 37). This corresponds to a technique of divide-and-conquer algorithms called memoization, i.e. identifying and saving the solutions to overlapping subproblems resulting from the branched recursion. Algorithm 2 starts by checking whether a previously computed solution for e already exists in the cache (cf. line 6). After each run of Algorithm 2, the cache is cleared (cf. Algorithm 1, line 7).

Algorithm 1 Compute the GIMME match of a (line or closed line) location, given as a list of source edges in a source map, in a destination map, and return it as a list of destination edges

Precondition: e is a list of edges in the source map, and $t \in$ {LINE, CLOSED_LINE}

```
 1: function GIMMEMATCH($e, t$)
 2:     $m \leftarrow$GEOMETRYMATCHES($e$)
 3:     if $m$.SIZE $\neq$ $e$.SIZE then    ▷ A match is needed for every source edge, cf. (ii)
 4:         return NULL                     ▷ No match
 5:     end if
 6:     $S \leftarrow$SOLUTIONS($e, m$)      ▷ $S$ is a set of edge lists
 7:     CLEARCACHE()                        ▷ Clear the result cache, cf. Algorithm 2
 8:     if S.ISEMPTY() then
 9:         return NULL                     ▷ No match
10:     else if $t$ = LINE then
11:         $s \leftarrow$LARGESTADMISSIBLESOLUTION($e, m, S$)
12:     else                                ▷ $t$ = CLOSED_LINE
13:         $s \leftarrow$FIRSTCLOSEDLINESOLUTION($S$)
14:     end if
15:     return $s$                          ▷ Solution $s$ is a list of edges
16: end function
```

3.2 Further remarks

This section gives further implementation details and discusses the requirements regarding digital road maps.

In Algorithm 1, line 3 the result of the GM algorithm (cf. Section 2.3.4) is checked for completeness (cf. (ii)), i.e. GIMMEMATCH returns a null value if there is a source edge for which GM could not return a candidate list. Notice that candidate lists returned by GM are never empty, but rather the entry for the respective source edge is missing in the returned associative array.

Further, it is of note that the algorithm adds the current potential solution for the head if no solution for the tail can be found (cf. Algorithm 2, lines 24 and 25): this occurs at the boundary case of the recursion, and whenever the candidate edges for the tail have already been “consumed” by matches for the first source edges. As long as for every edge of the tail at least one of their candidate edges has been used, this might still be a valid solution (cf. iii). Therefore, GIMME adds it tentatively in Algorithm 2, line 25. On the other hand, some of these smaller potential solutions may actually not be valid solutions. The current modus operandi of GIMME is to sort them out later by choosing a *largest* admissible solution as the final solution for the source route (cf. Algorithm 1, line 11). As had already been mentioned,

Algorithm 2 Construct all possible matches for a source route from the map of candidate lists, return them as a set of edge lists

Precondition: e is a list of edges in the source map, m is an associative array mapping the source edges in e to lists of candidate edges in the destination map

```
function SOLUTIONS($e, m$)
    if $e$.ISEMPTY() then
        return $\emptyset$    ▷ Boundary case of the recursion
    end if
    if ISCACHED($e$) then
        return CACHELOOKUP($e$)    ▷ Return cached result
    end if
    $f \leftarrow e$.GET(0)    ▷ $f$: first source edge
    $M \leftarrow$SET($m$.GET($f$))    ▷ $M$: set of matching candidates for $f$
    $n \leftarrow e$.SIZE()
    $X \leftarrow \emptyset$    ▷ $X$: set of extended solutions, initially empty
    $P \leftarrow$CONSTRUCTPOWERSET($M$)    ▷ $P = 2^M$, $M$: set of matching candidates for $f$
    for all $S \in P$ do    ▷ $S$ loops over all subsets of $P$
        $D \leftarrow$CONNECTEDPERMUTATIONS($S$)    ▷ $D$: set of directed paths, cf. Algorithm 3
        if $D$.ISEMPTY() then
            continue
        end if
        for all $d \in D$ do    ▷ $d$ loops over all directed paths of $D$, holds head solution
            $T \leftarrow \emptyset$    ▷ $T$: set of solutions for the tail of $e$, initially empty
            if $n > 1$ then    ▷ Tests whether passed source route has a second edge
                $t \leftarrow e$.SUBLIST($1, n$)    ▷ $t$: tail of $e$
                $T \leftarrow$SOLUTIONS($t, m$)    ▷ Recursive call, $T$ is a set of edge lists
            end if
            if $T$.ISEMPTY() then    ▷ Tests whether there are no solutions for the tail
                $X \leftarrow X \cup \{d\}$    ▷ Add edge list $d$ (current head solution) to the set $X$
            else
                for all $s \in T$ do    ▷ $s$ loops over all solutions in $T$
                    if $s$.HASPREFIX($d$) then    ▷ Tests whether $s$ subsumes head solution $d$
                        $X \leftarrow X \cup \{s\}$    ▷ Add edge list $s$ to the set $X$
                    end if
                end for
                $L \leftarrow$LEFTEXTENSIONS($d, T$)    ▷ $L$ is a set of edge lists, cf. Algorithm 4
                $X \leftarrow X \cup L$    ▷ Add all newly found left-extended paths to $X$
            end if
        end for
    end for
    CACHEINSERT($e, X$)
    return $X$    ▷ $X$: set of edge lists (each of them representing a possible match)
end function
```

a largest admissible solution is preferable in any case. This is because such a solution is more likely to cover the whole source route (i.e. to not miss any part of it). Further, it is not a problem in practice if it is longer than the source route since then offsets from the start of the first target edge and/or from the end of the last target edge (i.e. offsets trimming the path down to the size of the location path, cf. Section 2.2) can be provided together with the matched route.

Moreover, these offsets are already calculated by the GM algorithm ([26]), and henceforth it is sufficient to simply pass them from the matching result of GM to GIMME (see also the worked example in Section 3.4). This is straightforward, and no further instructive code for this has been included in the pseudo code. As for closed line solutions, no offsets need to be calculated since the routes are circular. In this case, Algorithm 1 only needs to look for the first potential solution which meets the requirement of representing a cyclic path (cf. Algorithm 1, line 13).

Algorithm 3 Construct all connected permutations for a set of edges, i.e. all permutations corresponding to a directed path in the underlying directed graph, and return the corresponding directed paths as a set of edge lists

Precondition: S is a set of edges

```
 1: function CONNECTEDPERMUTATIONS($S$)
 2:     if $S$.ISEMPTY() then
 3:         return $\emptyset$
 4:     end if
 5:     $X \leftarrow \emptyset$    ▷ $X$: set of extended solutions, initially empty
 6:     for all $e \in S$ do
 7:         $R \leftarrow S \setminus \{e\}$    ▷ $R$: remaining set without edge $e$
 8:         $D \leftarrow$ NULL
 9:         if not $R$.ISEMPTY() then
10:             $D \leftarrow$ CONNECTEDPERMUTATIONS($R$)    ▷ Recursive call on remainder $R$
11:         end if
12:         if $D =$ NULL then    ▷ $D =$ NULL: call in line 10 has not been executed
13:             return $\{(e)\}$    ▷ Returning a set of lists, containing only list $(e)$
14:         else
15:             $L \leftarrow$ LEFTEXTENSIONS($(e), D$)    ▷ $L$ is a set of edge lists, cf. Algorithm 4
16:             $X \leftarrow X \cup L$    ▷ Add all newly found left-extended paths to $X$
17:         end if
18:     end for
19:     return $X$
20: end function
```

Algorithm 4 Let a directed path be represented by a list of directed edges. For each path in a given set of directed paths, try to extend it to the left by a given directed path (i.e., extend a path if the last edge of the given path is connected to the first edge of this path), and then return the set of all paths that could be extended (i.e., a set of edge lists)

Precondition: e is a list of edges, D is a set of edge lists, each such list forming a directed path in the underlying directed graph

```
function LEFTEXTENSIONS($e, D$)
    $X \leftarrow \emptyset$    ▷ $X$: set of extended paths, initially empty
    for all $d \in D$ do    ▷ $d$: edge list in $D$
        $l \leftarrow e$.GET($e$.SIZE() − 1)    ▷ $l$: last edge of $e$
        if ISCONNECTED($l, d$) then    ▷ Tests whether $l$ is connected to first edge of $d$
            $x \leftarrow d$.COPY()    ▷ A shallow copy (e.g. Java: clone()) is intended here
            $x$.PREPEND($e$)    ▷ Prepend all elements of $e$ (e.g. Java: addAll(0, $e$))
            $X \leftarrow X \cup \{x\}$    ▷ $X$: set of lists, i.e. of all (extended) paths
        end if
    end for
    return $X$
end function
```

The GM algorithm used by GIMME calculates distances between vertices, angles between edges, and the area of a mutual projection of edges. Moreover, GM calculates the difference between the functional road class (FRC) of the source edge and the FRC of a candidate edge, and then integrates this difference with a certain weight into a weighted sum for the candidate rating [26], see also the next section, Section 3.3. Therefore, GIMME requires the maps to have information annotated with the vertices and edges which allow for all these calculations, and in sufficient precision. These requirements are met if the coordinates of the road network nodes are stored as WGS84 coordinates (the accuracy should not be less than decamicrodegrees for each value), and if every road segment has a functional road class value indicating its importance in the network. Notice that these are actually less requirements than for OpenLR which also requires maps to have form of way (FOW) values for every road segment. Digital road maps from well-known vendors like TomTom/TeleAtlas, HERE/NAVTEQ, and the collaborative project OpenStreetMap comply with these requirements.

3.3 Dissimilarity threshold

While GM [26] computes a dissimilarity measure and then orders the candidates in the candidate list with respect to this measure, GIMME currently does not use this order (see Section 3.1). Nonetheless, GIMME uses GM to calculate the list of potential candidates, and GM's thresholds for exclusion of a candidate from the candidate list are briefly reviewed in the following.

Figure 1a exemplifies a pair of a source edge and a corresponding candidate edge. Figure 1b shows the angle α between the two edges, and their mutual projection. The length of the projection of the candidate to the source (i.e. reference) edge is called the reference length, shown in Fig. 1c together with the intersection of the areas confined to the edges and the perpendiculars of their mutual projection, respectively. The average distance a is defined as the ratio $\frac{A}{l}$ of this area A and the reference length l.

Moreover, two compound dissimilarity measures are calculated. The first is purely geometrical and the second also integrates Δ_{FRC}, the absolute difference of the functional road class (FRC) value between the candidate edge and the source edge. A candidate is excluded if at least one of its dissimilarity measures is above a numerical threshold:

$$\begin{aligned} D &= a + a \cdot \alpha + a \\ D' &= a + a \cdot \alpha + \frac{a \cdot \Delta_{\text{FRC}}}{2} \end{aligned}$$

The angle α is assumed to be given in rad. The thresholds for inclusion of a candidate in the candidate list are:

a) Maximum average distance a: 20 m
b) Maximum angle α: 40° $\widehat{=}$ 0.698 rad
c) Minimum mutual projection length: 3 m
d) Maximum for D: 30
e) Maximum for D': 40

The thresholds and weights have been calibrated during the evaluation of various digital road maps from different vendors with respect to their geometrical features. During this evaluation, all edges of different pairs of maps of Brunswick, Germany, and as a second test case also of Hanover, Germany, have been matched with GM, respectively [26]. I.a. the average distance between homologous edges can be significantly more than 10 m, due to varying accuracy of the digitalisation from analogue map data, and due to different manufacturing methods [26]. Moreover, in one map a road may be represented by dual carriageways with a physical barrier separating the carriageways, but as a bidirectional road in another. GM accounts for this by tolerating the resulting larger angular differences between homologous roads ([26], see also Fig. 14 in Section 4.2.5).

3.4 A real-world worked example

This section demonstrates the proposed algorithm GIMME with a small real-world example. GIMME is used to match a stretch of road in a TeleAtlas map of Potsdam, Germany

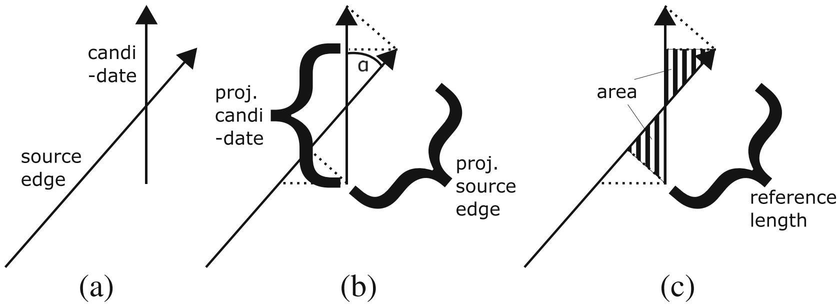


Fig. 1 Aspects of geometry-based dissimilarity thresholds

(Friedrich-List-Straße between Nuthestraße and Rudolf-Breitscheid-Straße), consisting of two road segments, to a NAVTEQ map of Potsdam (see Fig. 2).

We denote the two road segments (or edges in the underlying graph, respectively) by a and b (see the upper TeleAtlas part in Fig. 2). The driving direction is from a to b, and b is an outgoing edge of a. In the following, it is described how GIMME matches the short route (a, b) to two road segments (edges) in the NAVTEQ map, denoted x and y. The driving direction here is from x to y, and y is an outgoing edge of x. Candidate edge x has a mutual projection length above the threshold of 3 m with source edge a (the remaining dissimilarity measures given in Section 3.3 do not exceed the respective maximum values, respectively), but not with source edge b. In contrast, candidate edge y has a mutual projection length above the threshold with *both* source edges a and b (and also the remaining dissimilarity measures do not exceed the respective maximum values, respectively).

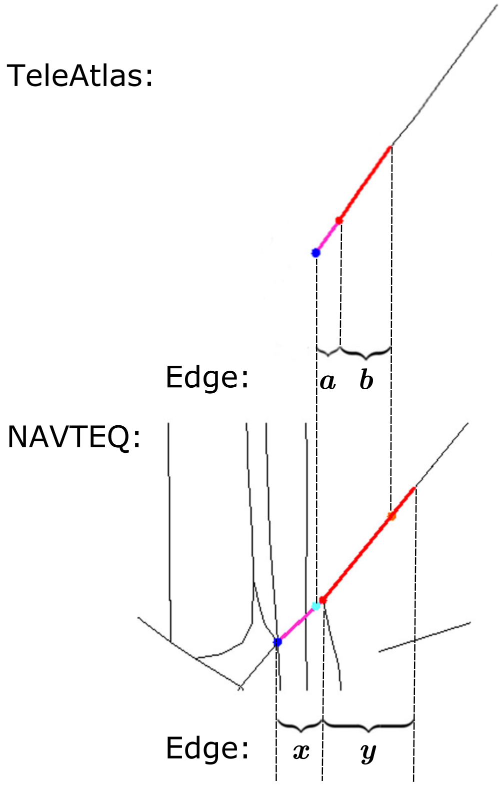


Fig. 2 A worked example: matching a stretch of road in Potsdam, Germany (Friedrich-List-Straße between Nuthestraße and Rudolf-Breitscheid-Straße) with GIMME

Therefore, besides the longest (and therefore best) target route (x, y), also a route consisting of only the candidate edge y (i.e. the route (y)) will be considered as a possible solution during the matching process of GIMME.

GIMME is invoked by a call to GIMMEMATCH with parameters $e = (a, b)$ (the list of edges), and t = LINE (the type of the location, i.e. a linear route). After a call to the GM algorithm with argument e, map m is assigned to an associative array: $m = \{a : (x, y), b : (y)\}$. Here, x, y denote the candidate edges identified by the GM algorithm (see the lower NAVTEQ part in Fig. 2). NAVTEQ-edges x and y are connected in driving direction, i.e. y is an outgoing edge of x. Map m maps edge a to the list of candidates (x, y), and edge b to (y), that is, to a singleton with only one candidate (NAVTEQ-edge y). No details of operation of the GM algorithm are given here (see Section 2.3.4 and [26] for more details about GM). In line 6 of Algorithm 1 (GIMMEMATCH), Algorithm 2 (SOLUTIONS) is called with parameters e and m.

This is a recursive algorithm, and the shallowest calling level has been invoked. This calling level will be referred to as level 0. During the run of the algorithm, level 0 will recursively call SOLUTIONS, thereby invoking a deeper calling level, which will be referred to as level 1, and so on.

In **level 0**, solutions for head and tail of e, the passed source route, are considered subsequently. The code starts by addressing the head: for this purpose, M is assigned to the set of candidates for the first element of $e = (a, b)$, i.e. for a. That is, according to map m, M is assigned to $\{x, y\}$. Next, the power set of M is computed in line 12 of Algorithm 2, and assigned to P, i.e. $P = \{\emptyset, \{x\}, \{y\}, \{x, y\}\}$.

Now S loops over all subsets of P, starting by $S = \emptyset$. Here, nothing has to be done (see lines 15 - 17). Next, $S = \{x\}$, and via a call to CONNECTEDPERMUTATIONS, D is assigned to the set of all directed paths which are constructable with S, that is, $D = \{(x)\}$.

Now d loops over all directed paths in D. In other words, d loops over all possible solutions for the head of the route e, constituted by candidates of the currently considered subset S. There is only one iteration with $d = (x)$. The tail of list $e = (a, b)$ passed to level 0 is assigned to t, i.e. $t = (b)$. To address the tail, SOLUTIONS is called recursively on the tail in line 22, that is, t is passed as the formal parameter $e = (b)$, the same map m is passed, and level 1 is invoked.

In **level 1**, M is assigned to $\{y\}$ since the first element of $e = (b)$ is b and since m maps b to (y) . Next, the power set P is computed as $P = \{\emptyset, \{y\}\}$, and again S loops over all subsets of P, starting by $S = \emptyset$. Again, nothing has to be done here. In the next iteration, $S = \{y\}$, and D is calculated as $D = \{(y)\}$ by a call to CONNECTEDPERMUTATIONS.

Now there is again only one iteration for loop variable d with $d = (y)$. The tail of list $e = (b)$ is the empty list, and therefore no recursion on the tail takes place (see line 20). Therefore, the current solution for the head of the level 1 route (i.e. $d = (y)$) is added in line 24 to the tentative set of solutions X (which had initially been initialized as the empty set), i.e. $X = \{(y)\}$. Another side effect is that lines 26 to 34 are not executed: since no solutions have been found for the tail of the level 1 route, there is no need to check for possible connections between the current head solution ($d = (y)$) and tail solutions. Consequently, no more solutions are found on this level, and level 1 returns $X = \{(y)\}$ as the set of solutions found for $e = (b)$. Notice that before returning to level 0 this solution is inserted into the cache with the key (b) (see line 37).

Execution returns to **level 0**, back to the iteration with $S = \{x\}$, and to the (one) iteration of the inner loop in line 18 with $d = (x)$. Lines 26 to 34 first check whether the current head solution is a prefix of a tail solution (which is not the case for head solution $d = (x)$ and tail solutions $T = \{(y)\}$), and then call subroutine LEFTEXTENSIONS in line 32 (with parameters $d = (x)$ and $T = \{(y)\}$) which subsequently attempts to topologically connect the current solution for the head to solutions for the tail. Since edge y is an outgoing edge of edge x in driving direction (see Fig. 2), this works for $d = (x)$ and the tail solution (y), and hence the new solution (x, y) is added to the tentative set of solutions X. Since X had been initialized as the empty set, it is $X = \{(x, y)\}$.

In level 0, execution continues to loop over all subsets of P, i.e $S = \{y\}$, followed by the assignments $D = \{(y)\}$, $d = (y)$, and again the recursive call on the tail of $e = (a, b)$, i.e. on $t = (b)$. In line 6, **level 1** of SOLUTIONS immediately returns $X = \{(y)\}$ since this solution had been inserted into the cache with the key (b) in a previous call on level 1 (see above).

Back at **level 0**, the returned solution is assigned to T, i.e. $T = \{(y)\}$. Since T is not empty, the algorithm now checks whether the current head solution is a prefix of a tail solution in line 28. This is the case since $d = (y)$ is a prefix of $(y) \ni \{(y)\} = T$. Therefore, (y) is added to $X = \{(x, y)\}$, and we have $X = \{(x, y), (y)\}$. The subsequent call to LEFTEXTENSIONS in line 32 on the arguments $d = (y)$ and $T = \{(y)\}$ yields the empty set since there is no topological connection between the last edge of the current head solution, i.e. between y, and a tail solution in $T = \{(y)\}$ (see Fig. 2). Thus, at the end of the iteration with $D = \{(y)\}$ and $d = (y)$, X remains unchanged, i.e. $X = \{(x, y), (y)\}$.

The next considered subset of P is $S = \{x, y\}$, and thus we have the assignments $D = \{(x, y)\}$ and $d = (x, y)$, respectively, followed by the recursive call on the tail of $e = (a, b)$, i.e. on $t = (b)$. Like before, **level 1** of SOLUTIONS immediately returns $X = \{(y)\}$ from the cache.

Back at **level 0**, the returned solution is assigned to T, i.e. $T = \{(y)\}$. Firstly, $d = (x, y)$ is not a prefix of a solution in $T = \{(y)\}$. Secondly, there is no topological connection between the last edge of the current head solution, i.e. between y, and a tail solution in $T = \{(y)\}$ (see Fig. 2). Consequently, X remains unchanged, and we have $X = \{(x, y), (y)\}$ as the final set of solutions for $e = (a, b)$ on level 0. This set is returned to Algorithm 1 (GIMMEMATCH), and assigned to S.

Since S is non-empty, and since the type of location is $T =$ LINE, LARGESTADMISSIBLESOLUTION is called with parameters $e = (a, b)$, the same map m as before, and with $S = \{(x, y), (y)\}$. The largest admissible solution is (x, y), and this solution is returned as the final match for (a, b), together with a positive offset of 27 meters and a negative offset of 18 meters. The offsets had been calculated before by GM for each candidate edge, and are passed through additional attributes in m (which is a technical detail that, for brevity, has not been addressed in the instructive code of Algorithm 1).

Figure 3 summarises how the initial problem of the worked example is divided into smaller subproblems. This results in a total of $|e| = n = 2$ recursive calls of Algorithm 2 (SOLUTIONS) on the subproblems (tails), returning a non-empty set of solutions which has actually been computed during the respective call, i.e. which is not directly returned from the result cache. In this paper, we call such calls *relevant* since they are "doing the actual work": more precisely, a recursive call is called relevant, if (i) it is invoked during the matching process of a non-empty and successfully matched source route, and if (ii) it does not return a result found in the result cache, and if (iii) it returns a non-empty list of solutions.

The initial source route and the tails on which Algorithm 2 has been invoked by a relevant call, are marked with left arrows pointing to them.

It is of note that always n relevant recursive calls of Algorithm 2 are executed if a source route is matched successfully (i.e. this is not a property specific to only the

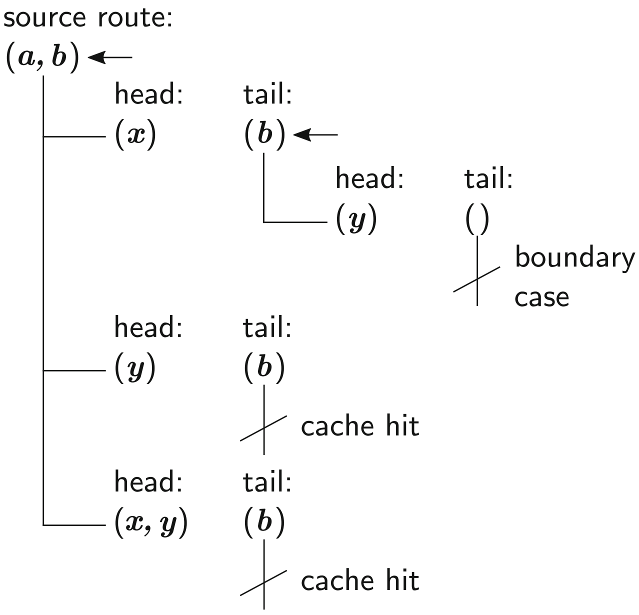


Fig. 3 Dividing the initial problem and recursive calls on the subproblems for the worked example in Fig. 2

demonstrated example). This is because for $n > 0$ the initial call on $e = (e_1, \ldots, e_n)$ always calls the algorithm recursively on $(e_2, \ldots, e_n)$ for every candidate solution for (e_1), and the first such call already induces a chain of $n - 1$ subsequent recursive calls on increasingly short tails (i.e. calls on $(e_2, \ldots, e_n), \ldots, (e_n)$. Since e is assumed to be successfully matched, every such call must return a non-empty list of solutions, and therefore every such list of solutions is inserted into the result cache with the respective key. In particular, the list of solutions for the subproblem $(e_2, \ldots, e_n)$ is cached with this subproblem as the key. Therefore, any further recursive call of the initial call on $(e_2, \ldots, e_n)$ is not relevant since the solution is immediately returned from the cache.

This property of always having n relevant recursive calls for a successful match will also play an important role in the discussion of the amortised average run time in Section 4.2.3.

3.5 Combining GIMME with a bandwidth-efficient method

Notice that GIMME needs access to *both* the source and the target map. This requirement is met whenever an omniscient matching centre with access to both maps can be used. If location references need to be transmitted from a sender to a receiver, the algorithm can compute the matches at the receiving side, using both maps. The input of GIMMEMATCH is essentially a list of edges. In the case of a matching centre, there is usually a common main memory for sender and receiver instances, and then this list can be represented as e.g. a list of instance variables pointing to edge objects in main memory (then assuming that the maps have been loaded into an object-oriented structure of vertex and edge objects, see e.g. [13]).

In the case that sender and receiver are not running on the same machine, a list of edge IDs for the respective source map format can be used. Usually, transmission of this ID list will require (much) more bandwidth than e.g. transmitting the small set of intermediate LRPs used by an OpenLR reference to uniquely identify the location (where these LRPs represent a concatenation of shortest-paths covering it, see Section 2.2). Indeed this is prohibitive to a direct usage of GIMME if bandwidth is restricted for one or more transmission lines between sender and receiver. Fortunately, it is possible to obtain a compact format for the descriptive data before transmission. For this purpose, GIMME is combined with a bandwidth-efficient dynamic location referencing method like e.g. OpenLR. More precisely:

It is well-known that dynamic location referencing protocols usually work perfectly (i.e. with an accuracy of 100%) if source and target map coincide (e.g. this is the case for OpenLR). This can be exploited for avoiding the transmission of the whole list of edge IDs over a bandwidth-restricted line: instead, e.g. a compact OpenLR reference is created in the source map at the sender side, and transmitted over the respective line. The receiver side holds the same source map, and so the OpenLR reference can be correctly decoded in it. Thereby the list of edges of the encoded route is recreated, represented as e.g. a list of instance variables pointing to edge objects. This list can now be passed to GIMME (e.g., as an instance variable via the internal main memory of the receiving server) without having to respect any further bandwidth requirements, and usually without any decrease in accuracy.

A certain drawback is the increased run time of the decoder, since every location needs to be decoded two times, first with e.g. OpenLR and second with GIMME. Notice that the sender side runs e.g. an OpenLR encoder, and the receiver side runs e.g. an OpenLR decoder and GIMME. Moreover, the receiver side running GIMME needs access to (an identical copy of) the source map, and to the destination map.

4 Evaluation

This section describes the results of an evaluation of the proposed algorithm GIMME. In a first subsection, Section 4.1, evaluation measures are defined, which correspond to well-known measures in e.g. binary classification in Machine Learning. The second subsection, Section 4.2, gives the results of experiments in terms of the introduced evaluation measures. For the experiments, the proposed algorithm GIMME and OpenLR have been used to match

- short line locations (linear routes) consisting of up to 5 road segments
- closed line locations (i.e., tours or circular routes) consisting of up to 69 road segments

between a TeleAtlas and a NAVTEQ map of the city of Potsdam, Germany.

Run time measurements give a comparison of run times for GIMME and OpenLR, and demonstrate the linear relation between route length and run time of GIMME. This linear relation then is subject to a more formal discussion of the amortised average run time in Section 4.2.3. Finally, the section discusses the results of previous experiments with RNM algorithms, and compares them to the obtained results for GIMME.

The outlined experimental setup complies with the use of GIMME for dynamic location referencing (DLR), which has been the main use case in several projects of German Aerospace Center: in the projects MobiLind, KeepMoving, and I.MoVe, traffic information like e.g. the information about congested stretches of road, or the spatial extent of a whole congestion area needed to be processed as incoming OpenLR-references like e.g. OpenLR line locations or closed line locations (please recall that closed line locations are *area locations* which can be used to describe two-dimensional parts of the surface of the earth, see Section 2.1). During the projects, these references had been encoded in a TeleAtlas map, but the used process chain required to decode them in a NAVTEQ map.

Moreover, the design of the setup is such that it is very similar to the setups in previous works in the context of DLR. This has the advantage of making our results comparable to previously reported results.

Notice that protocols for DLR usually encode the whole stretch of road or the whole area in question in one single location reference, which they transmit in one piece from sender to receiver. That means, a location is not split into e.g. multiple individual edges, and then distributed across a respective number of location references. This would mean unnecessary overhead, contradicting the aim of bandwidth reduction: e.g. OpenLR achieves a compact format by encoding linear or circular routes as sequences of intermediate location reference points (LRPs, see Section 2.2). Their number is usually much smaller than the number of edges. Moreover, every message has a header, and thus sending multiple messages would mean increasing the overhead.

In the scenario of an omniscient matching centre with access to both the source and the target map, there are no restrictions for the bandwidth. Nonetheless it is still advantageous to match complete locations instead of matching single edges: matching complete (linear or circular) routes with GIMME (cf. Section 3.1) instead of matching individual edges can yield more accurate results. This is because the algorithm keeps multiple matching candidates per edge, and since a best combination of candidates for all route edges is constructed. That way, GIMME is capable of correcting a wrong choice subsequently, until a best admissible (i.a. directed) path in the target map has been found (for more details, see Section 4.2.4).

It is of note that, in the context of this paper, a successful match always means a *complete* match, i.e. partial matches are not of interest and are considered as a failed match. This complies with the usual approach in DLR: matching failures, i.e. unmatched location references can easily be suppressed or replaced at the application level, e.g. a traffic-monitoring application can always drop an unmatched reference to a congested stretch of road, or perhaps replace it by a less accurate, more general textual warning. In contrast, basing traffic information on a partial match holds the risk of providing wrong information to the user during e.g. route guidance or navigation. This is the main reason why DLR-protocols like e.g. OpenLR usually discard partial matches and simply report them as a decoding error.

4.1 Evaluation measures

Let n be the total number of locations as sent in a test. For the scope of this test, let n_{p} be the number of positives, i.e. the number of matched routes, and let n_{n} be the number of negatives, i.e. the number of routes which could not be matched. Let n_{tp} denote the number of true positives, i.e. the number of correctly matched routes where source and target route were homologous. Let n_{fp} denote the number of false positives, i.e. the number of matched routes where source and target route were not homologous. Further, let n_{tn} be the number of true negatives, i.e. the number of routes truthfully identified as not having a homologous counterpart in the target map, and let n_{fn} be the number of false negatives, i.e. the number of routes which could not be matched although they have a homologous counterpart in the target map. For a real-world evaluation, these numbers are subject to the following constraints:

$$\begin{aligned} n = n_{\mathrm{p}} + n_{\mathrm{n}} &> 0 \\ n_{\mathrm{p}} \geq n_{\mathrm{tp}} &\geq 0 \\ n_{\mathrm{n}} \geq n_{\mathrm{tn}} &\geq 0 \end{aligned}$$

Further, let $n_{\mathrm{t}} = n_{\mathrm{tp}} + n_{\mathrm{tn}}$. The percentage hit rate q_{hit}, success rate q_{success}, and error detection rate $q_{\mathrm{error\_detection}}$ are defined by

$$q_{\mathrm{hit}} = \frac{n_{\mathrm{t}}}{n} \cdot 100\% \qquad (1)$$

$$q_{\text{success}} = \frac{n_{\text{tp}}}{n_{\text{p}}} \cdot 100\% \quad (2)$$

$$q_{\text{error\_detection}} = \frac{n_{\text{tn}}}{n_{\text{n}}} \cdot 100\% \quad (3)$$

respectively, where $q_{\text{hit}} = 100\%$ if $n = 0$ and q_{hit} is undefined for $n < 0$, and analogously for q_{success} and $q_{\text{error\_detection}}$.

Notice that the hit rate, success rate, and the error detection rate correspond to the well-known evaluation measures "accuracy", "precision", and "negative predictive value" for a confusion matrix in e.g. Machine Learning, respectively.

The hit rate is combining the two aspects of good performance of a dynamic location referencing method, namely correct location decoding and truthful detection of absence of a homologous counterpart in the target map. This however introduces a subtle dependency on the particular maps, together with the tested locations, e.g. on the ratio of true positives to true negatives in a particular constellation. Giving both success and error detection rates separately instead has also the advantage of providing more detailed information about the two different aspects of quality.

4.2 Experimental results

This section describes the results of experiments with GIMME and OpenLR, and also gives a comparison to previous RNM approaches. In detail, the section is divided into subsections as follows: Firstly, Section 4.2.1 gives the results of experiments that have been conducted with OpenLR, with a hybrid of OpenLR and GIMME, and with GIMME standalone, applied to randomly generated line and closed line locations (in contrast to AGORA-C, a reference implementation of OpenLR is publicly available [31]). Secondly, Section 4.2.2 gives a comparison of run times for GIMME and OpenLR, and demonstrates the linear relation between route length and run time of GIMME. This linear relation then is subject to a more formal discussion in Section 4.2.3, which considers the amortised average run time. Next, Section 4.2.4 discusses the results of previous experiments with RNM algorithms, and compares them to the obtained results for GIMME. Finally, Section 4.2.5 gives depicted examples for successful matches of line and closed line locations as resulting from GIMME, and also discusses a situation where GIMME fails to find the correct match, and how to remedy it.

4.2.1 Quantitative comparison of GIMME and OpenLR with respect to evaluation measures

In previous experiments described in the literature, the results of applying AGORA-C and OpenLR to line locations have been assessed [17, 29, 43]: In [29], i.a. 1,000 randomly selected line locations in The Netherlands have been encoded in a TeleAtlas map, and then decoded in a NAVTEQ map. In [43], i.a. 881 randomly selected line locations in The Netherlands have been processed in the opposite direction, i.e. encoded in a NAVTEQ map, and then decoded in a TeleAtlas map. These locations each consisted of a set of one to five connected edges forming a path in the road network.

In order to arrive at comparable results, an experimental setup similar to the setups in the aforementioned previous works seemed preferable. OpenLR has the advantage of a publicly available reference implementation, and therefore it was decided to compare GIMME with OpenLR. To be in line with the approach in [29], encoding was done in a TeleAtlas map, and decoding in a NAVTEQ map. Firstly, 1,000 short routes in a TeleAtlas map of Potsdam, Germany (TA 2012.06), consisting of 1 to 5 road segments like in [43] have been created randomly. They comprised of 2.84 edges on average, and have been created without any further structural restriction. Consequently, the generated routes were allowed to intersect themselves, and also to be circular as a special case of a linear route. Nonetheless, the vast majority of generated routes were non-circular and did not intersect themselves. All generated routes have a start node and an end node, and are lists of edges representing road segments which are connected in such a way, that a vehicle can drive on them in the driving direction, that is, from start to end.

Secondly, 1,000 closed line locations have been generated randomly in the same TeleAtlas map for Potsdam. Essentially, they are linear routes where start and end node coincide. In effect, closed lines are circular, and form a special case of linear routes. Again, closed lines are lists of edges representing road segments, and a vehicle can drive on them in the driving direction from start to end. Different from a non-circular route, a vehicle can also continue to drive past the end, thereby starting a next "round", and so on. They have been created such that a location boundary does not intersect itself, and such that the locations are as small as possible: that way, the experiments could be conducted under conditions comparable to those for the aforementioned experiments (e.g., [29, 43]). More precisely, the generated closed line locations comprised of 18.55 edges on average, the number of connected edges forming the respective closed paths in the road network ranged from 2 to 69. Notice that in order to generate such a large sample of 1,000 randomly generated closed line locations, more than the maximum of five connected edges as in the evaluation of [43] had to be allowed for: this is simply because such a large number of distinct short closed lines does not exist in the road network of Potsdam, Germany.

All of the 1,000 generated random line locations and all of the 1,000 closed line locations have been encoded successfully. Three experiments have been conducted for each of the two location sets.

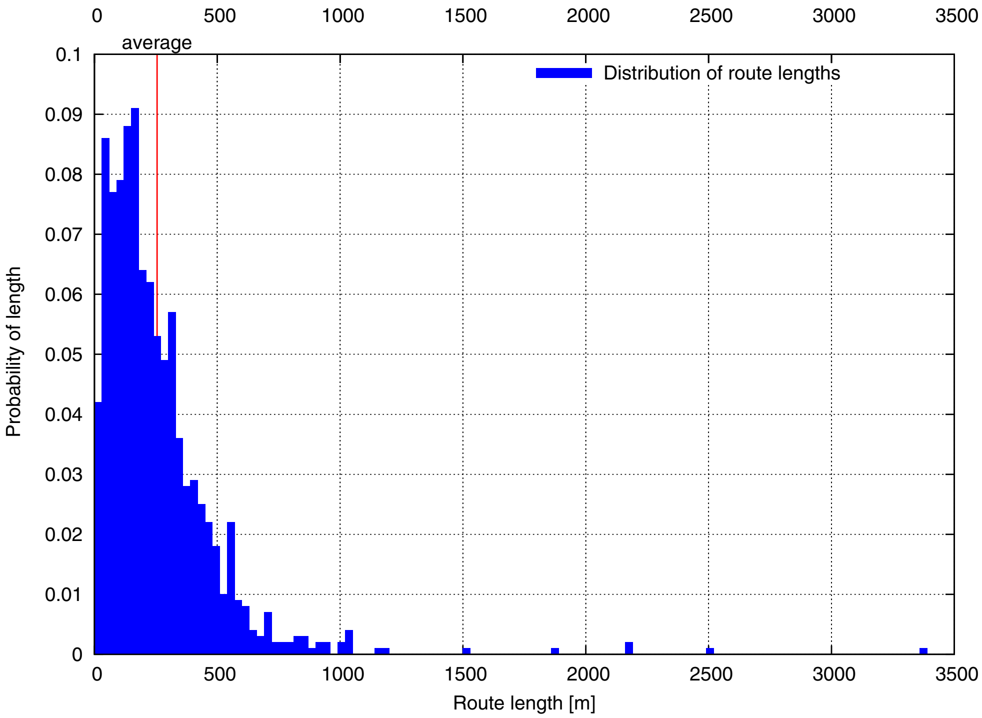


Fig. 4 Distribution of total location lengths in the first location set (line locations)

The lengths of the routes in the first set ranged from 5.0 m to 3,387.0 m (254.73 m on average). For the distribution of the route lengths in the generated sample, see Fig. 4 (for the cumulated distribution, see Fig. 5).

For the first experiment, each route in the first set was encoded with OpenLR 1.4.2 in the TeleAtlas map, and then decoded in the NAVTEQ map. The observed success and error detection rates were 91.9%, and 55.9%, respectively (for the definition of the two rates see Section 4.1). Both rates have been determined by manual inspection of the respective source and (potential) target routes using TomTom's MapViewer tool (see Fig. 6).

For the second experiment, GIMME has been applied to all routes of the first set which OpenLR had not been able

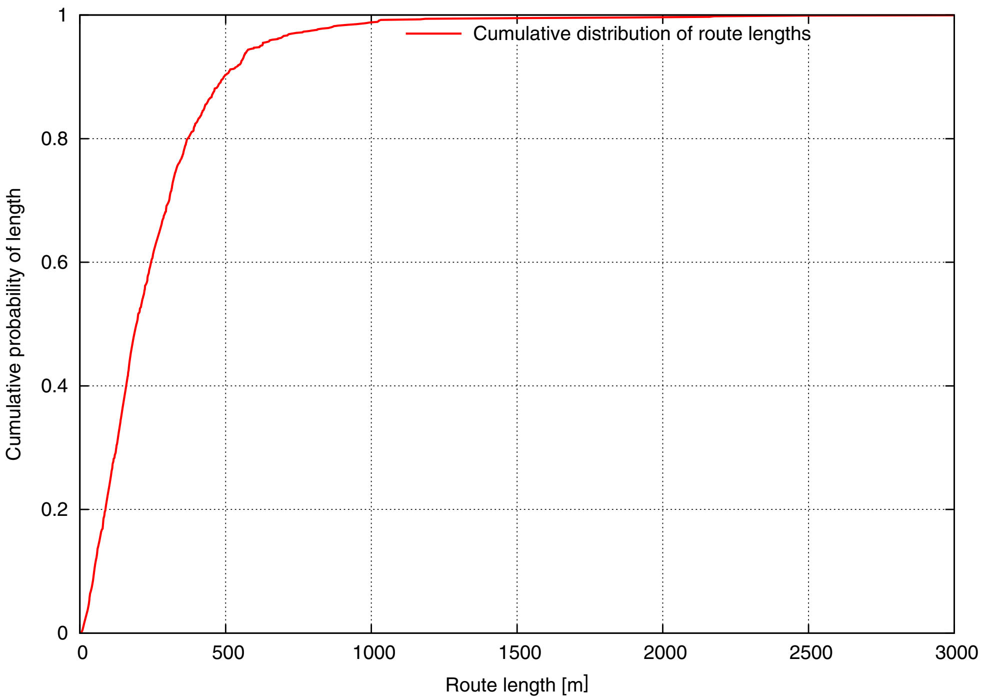
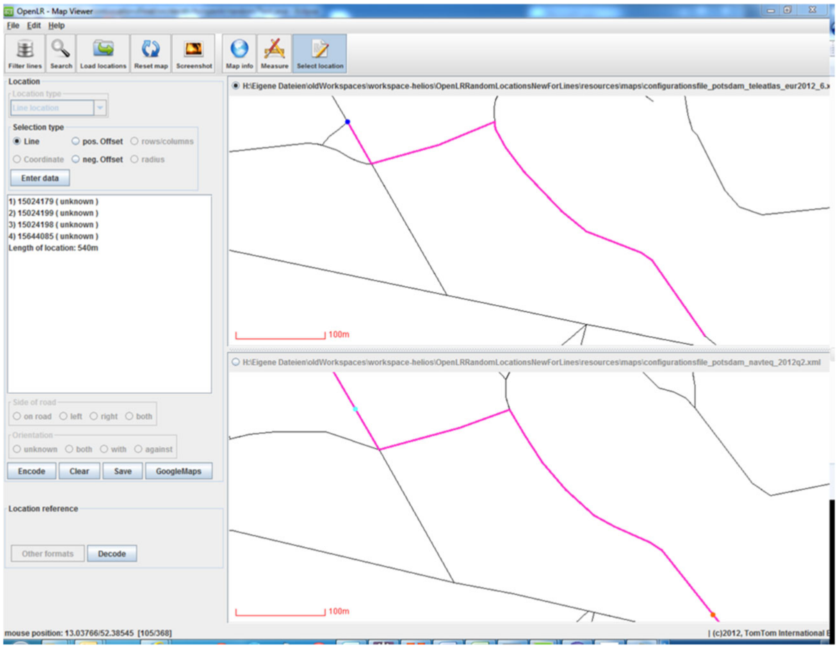


Fig. 5 Cumulative distribution of route lengths in the first location set (line locations), for lengths between 0 to 3,000 m



OpenLR - Map Viewer

Selection type: ● Line

1) 15024179 (unknown)
2) 15024199 (unknown)
3) 15024198 (unknown)
4) 15644085 (unknown)
Length of location: 540m

● H:\Eigene Dateien\oldWorkspaces\workspace-helios\OpenLRRandomLocationsNewForLines\resources\maps\configurationsfile_potsdam_teleatlas_eur2012_6.x

○ H:\Eigene Dateien\oldWorkspaces\workspace-helios\OpenLRRandomLocationsNewForLines\resources\maps\configurationsfile_potsdam_navteq_2012q2.xml

mouse position: 13.03766/52.38545 [105/368]

| (c)2012, TomTom International B

Fig. 6 Comparison of matched routes in a TeleAtlas map (*top*) and a NAVTEQ map (*bottom*) of Potsdam, Germany

to decode in the target map (i.e., a hybrid approach of first using OpenLR and then GIMME has been applied). For this purpose, GIMME was given full access to both the source and target map. In contrast, OpenLR as a map-agnostic dynamic location referencing protocol has only access to the source map at the sender side, and to the target map on the receiver side. This is sufficient for OpenLR since OpenLR only transfers intermediate location referencing points as map-independent WGS84 positions, in order to limit the amount of descriptive data (cf. Section 2.2).

As a result, the hybrid approach decoded 149 more routes, and all of them correctly (which has been validated by manual inspection). This is an increase in correctly decoded routes of 31.2%. As a consequence, the success and the error detection rate of the hybrid approach was 94.2% and 65.0%, respectively.

In the third experiment, GIMME was used as a stand-alone algorithm to find matches in the NAVTEQ map for the same set of 1,000 routes in the TeleAtlas map. This resulted in a success rate of 99.7%, and in an error detection rate of 69.0% (as determined by manual inspection). Compared to the previous run with OpenLR, 100 or 21.0% more routes could be decoded correctly by GIMME. When comparing to the previous run with the hybrid approach, the hybrid yielded 49 more correctly decoded routes than GIMME stand-alone.

To understand this, first note the following: although GIMME has a better error detection rate than OpenLR, it is still prone to false negatives, and even more so is OpenLR. However, since GIMME and OpenLR are very different methods, this does not necessarily mean that GIMME and OpenLR will falsely report negatives for the same locations. But rather GIMME might fail to decode a location which OpenLR decodes correctly, and (probably more frequently) it might be vice versa. And since the hybrid approach will report a negative (i.e., a failure to decode a location) only if *both* OpenLR and GIMME failed to decode the location, the frequency of successful (and correct) decodings will be higher than for GIMME stand-alone. This seems to be an advantage of the hybrid approach. However, both the success rate and the error detection rate lie between those for OpenLR and for GIMME. This may be due to the fact that the measured numbers n_{tp} and n_{tn} are influenced by the lower accuracy of OpenLR (cf. Eqs. 2 and 3).

The success and error detection rates resulting from the first three experiments are summarised in Table 1. For the definition of the success rate and the error detection rate, see Eqs. 2–3.

The second set of test locations consisted of 1,000 randomly generated closed line locations. The circuit lengths ranged from 271.0 m to 3,833.0 m (1,469.24 m on average). For the distribution of the circuit lengths in the generated sample, see Fig. 7 (for the cumulated distribution, see Fig. 8).

Table 1 Success and error detection rates for line locations

Method	success rate [%]	error detection rate [%]
OpenLR stand-alone	91.9	55.9
Hybrid OpenLR/GIMME	94.2	65.0
GIMME stand-alone	99.7	69.0

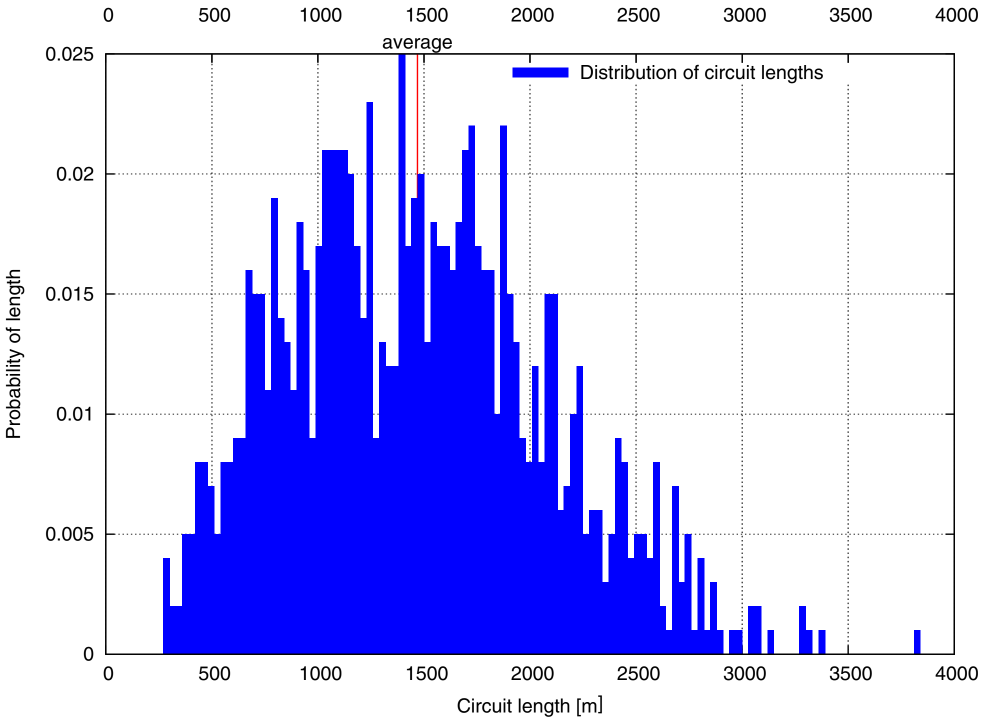


Fig. 7 Distribution of circuit lengths in the second location set (closed line locations)

The next three experiments were analogous to the first three, as the only difference was that they have been conducted with the closed line locations of the second set instead. In detail: for the fourth experiment, the set of closed line locations has been encoded with OpenLR 1.4.2 in the TeleAtlas map, and then decoded in the NAVTEQ map. For the fifth experiment, the aforementioned hybrid approach of first using OpenLR and then GIMME has been applied to the set of closed line locations. As a result, GIMME decoded 182 more closed lines, and all but 3 of them correctly (which has been validated by manual inspection). This is an increase in correctly decoded closed lines of 44.1%. In the sixth experiment, GIMME was used as a stand-alone algorithm to find matches in the NAVTEQ map for the set

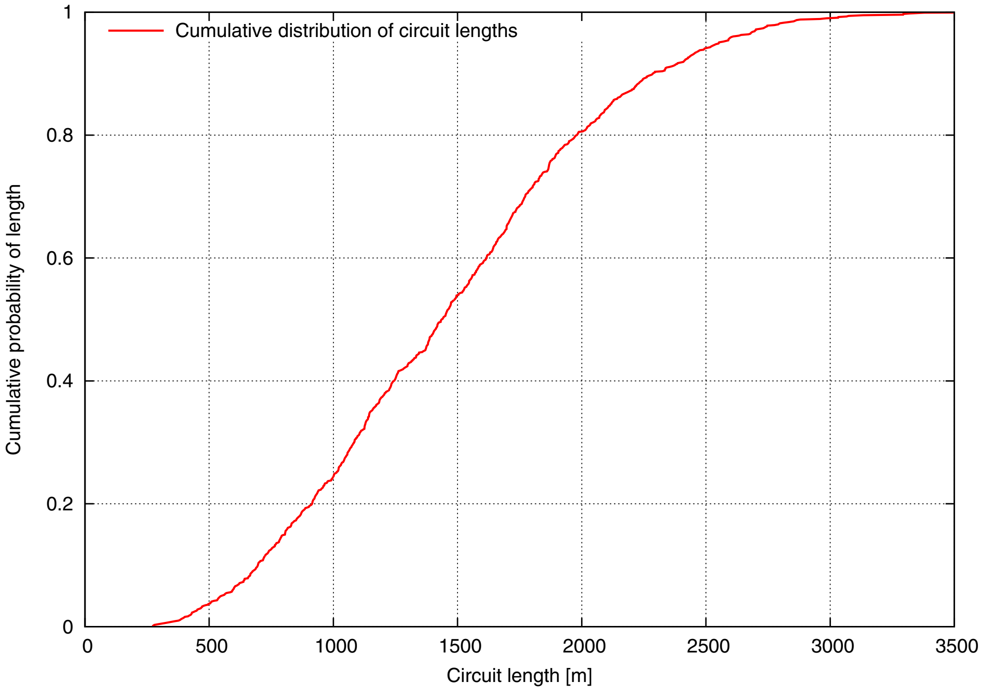


Fig. 8 Cumulative distribution of circuit lengths in the second location set (closed line locations), for lengths between 0 to 3,500 m

of 1,000 closed line locations in the TeleAtlas map. Compared to the previous run with OpenLR, 95 or 23.4% more closed lines could be decoded correctly by GIMME. That is, the hybrid approach showed properties analogous to those for line locations (i.e., the hybrid approach yields more correctly decoded closed lines than GIMME stand-alone, although the overall success rate of the hybrid approach is lower than for GIMME). The success and error detection rates for the last three experiments are given in Table 2.

To summarise: The results of the three methods are of different accuracy (in terms of success and error detection rate). The results of GIMME stand-alone have the highest accuracy, followed by the accuracy of the results of the hybrid approach, and the lowest accuracy has been achieved with OpenLR. Of note is that the hybrid approach yields the highest absolute number of correctly decoded locations. However, the use of OpenLR as part of it also introduces a higher number of false positives than GIMME stand-alone, which is the reason why the success rate of the hybrid approach is lower than for GIMME, though (cf. Eq. 2). The increase in accuracy as obtained by GIMME seems to be significantly larger for longer routes than for short routes. In other words, whereas OpenLR shows a clear degradation of accuracy with increasing length of the routes, there is only a small such degradation for GIMME.

4.2.2 Run time measurements

Firstly, the run times of processing the two sample sets (see Section 4.2.1), each with 1,000 locations, with the methods OpenLR, the hybrid approach, and GIMME (with and without caching), have been measured. The locations were loaded from hard disk, where they have been stored in XML format (one file per location). Since the respective loading times are significant, the results in Table 3 are given excluding and including the time for "I/O" (for brevity, "I/O" here refers to the loading times, plus the time for processing the location data by an XML parser), respectively. The experiments have been conducted on an IPC with an Intel Core i7-3770, 4x @3.40GHz CPU, with 16 GiB main memory, 4x 64 KiB L1 cache, 4x 256 KiB L2 cache, and 8 MiB L3 cache. OpenLR and GIMME were both built with Java, and have been running within the same execution environment and OS (Ubuntu Linux).

Table 2 Success and error detection rates for closed line locations

Method	success rate [%]	error detection rate [%]
OpenLR stand-alone	82.5	18.4
Hybrid OpenLR/GIMME	87.2	28.0
GIMME stand-alone	97.5	21.2

In the following, the given percentages all refer to the run times excluding the time spent for loading the location files from hard disk. While there is only a small benefit of the result caching (see e.g. Algorithm 2, line 6) when decoding only short line locations (only 3.9%), the gain in performance is large when decoding the longer closed line locations: GIMME without result caching required about 8.5 hours to decode the set of closed line locations, whereas it only took 361 seconds (not much more than 6 minutes) with the result cache, i.e. a speed-up of more than 85X. The hybrid approach has longer run times than OpenLR stand-alone (39.0% longer for line locations, and 115.7% longer for closed line locations). Since the hybrid approach additionally calls GIMME on those locations which OpenLR failed to decode, this is an expected result. For the short line locations, the run time of stand-alone GIMME (with caching) equals that of OpenLR (without using the cache, it was 8.5% longer). On the other hand, for the longer closed line locations, the run time of GIMME (with caching) was 95.1% larger than that of OpenLR. Hence, at least for longer routes, the significant increase in accuracy comes at the price of a longer run time.

Secondly, to investigate in the relation between route length and run time of GIMME, additional experiments have been conducted. Routes have been matched between the same two maps as in Section 4.2.1, consisting of different fixed numbers of distinct edges. These routes have been generated randomly, and besides the requirement of their fixed length, no further structural restriction has been applied. In each case 1,000 random routes consisting of 1, 5, 10, 15, 20 and 25 edges have been generated. The measured run times are depicted in Fig. 9 (green curve). The run time decreases with numbers of segments beyond 20, i.e. the respective curve is not monotonically increasing. Instead it is shaped like a parabola with a downwards opening.

The reason for this behaviour is that the probability of a positive match for a route is decreasing with the number of segments in it: the longer a considered source route is, the more unlikely it is that a homologous counterpart exists in the target map. In effect, the measured run time is that for a decreasing number of positives, plus the run time for the remaining number of negatives. The run time is dominated by the effort for the positives since on average, Algorithm 2 needs less time to identify a negative than a positive: for a negative, e.g. recursive calls on route tails (cf. Algorithm 2, line 22) tend to return less solutions, or even no solutions at all, saving effort in lines 26 - 34 of the algorithm. Moreover, for many of the non-existing routes, GIMME can identify a negative quickly in line 3 of Algorithm 1, that is, using the fact that the GM algorithm already identifies non-existence of a match (for single edges) by a quick proximity search. The contribution of these routes to the run time is negligible.

Table 3 Run times of the various methods for line and closed line locations

	Run times for 1,000 … line locations [s] excl. I/O	incl. I/O	closed line locations [s] excl. I/O	incl. I/O
OpenLR stand-alone	82	174	185	445
Hybrid OpenLR/GIMME w/ cache	114	206	399	659
GIMME stand-alone w/ cache	82	174	361	621
GIMME stand-alone w/o cache	89	181	30,692	30,952

The blue curve depicted in Fig. 9 shows a standardised run time, expressing how run time would evolve for 1,000 GIMME positives. It has been calculated from the measured run time by the following formula:

$$t_{\text{standardised}} = \frac{1,000}{n_{\text{p}}} \cdot t_{\text{measured\_for\_positives}} \tag{4}$$

where $t_{\text{measured\_for\_positives}}$ denotes the run time measured for the positives (i.e for the routes with a match in the target map) only.

The standardised run time is linearly increasing with the number of segments. The orange curve depicted in Fig. 9 shows a linear fit of the run time, using the function $y \approx 21.62 \cdot x + 5.06$.

During the previous experiment, also the run time of GIMME *without* the run time for calls of GM had been measured. It turned out that the run time of GM largely dominates the total run time of GIMME: the percentage of the run time for GM on the total run time of GIMME was 99.8%. To arrive at more reliable measurements for the run time of GIMME without the run time for GM, a second, similar series of experiments has been conducted with the same range of fixed numbers of route segments, but this time for 10,000 randomly generated routes each. The run times have been standardised like in Eq. 4, now using 10,000 in the enumerator instead of 1,000. They are shown in Fig. 10 (blue curve), together with a linear fit (dashed orange curve), using the function $y \approx 0.45 \cdot x + -0.24$.

As can be seen in Figs. 9 and 10, the relation of the (standardised) run time of GIMME to the route lengths is linear, regardless whether it is measured with or without the run times for calls of the GM algorithm.

In order to shed more light on this empirical result, the next section discusses the amortised average run time of Algorithm 2 (SOLUTIONS) in a more formal way. For this purpose, i.a. the relation of two quantities to the route length must be known. These quantities are being processed by Algorithm 2. The first is $|M|$, the number of candidates of a source edge, and the second is $|T|$, the number of returned solutions for the tail.

There is no obvious way to state useful (i.e. tight) theoretical bounds for $|M|$ and $|T|$. Instead, both quantities have been analysed empirically, and with randomly generated input: Firstly, for real-world urban road networks, the candidate lists M are usually rather short. In our experiments (see Section 4.2) their length did not exceed a maximum length of $C = 3$. Secondly, the *total* number of returned solutions for the tail during *all* recursive calls to Algorithm 2, has been observed for the first series of experiments with 1,000 lineear locations each, but such that only the positives contributed to this total count. In other words, locations that were not successfully matched (i.e. negatives) did not conribute to the total count, which is denoted with $T_{\text{observed\_for\_positives}}$ in the following. Then, a standardisation analogous to Eq. 4 has been applied (see Eq. 5).

$$T_{\text{standardised}} = \frac{1,000}{n_{\text{p}}} \cdot T_{\text{observed\_for\_positives}} \tag{5}$$

In Fig. 11, the blue curve shows that $T_{\text{observed\_for\_positives}}$ increases linearly with the route length. The dashed orange curve gives a linear fit using the function $y \approx 3559.02 \cdot x + -3517.75$.

To obtain the average number of returned solutions for the tail for one of the experiments with a fixed route length, an observed total must be divided by the number of locations, 1,000, and also by the respective route length n. The factor $\frac{1}{n}$ accounts for the fact that for every successfully matched source route exactly n relevant recursive calls of Algorithm 2 (SOLUTIONS) have been executed (for more details, see the discussion of Fig. 3 in the worked example in Section 3.4, and see Section 4.2.3). The mean $\overline{T}_{\text{standardised}}$ of the six average numbers for the respective fixed route lengths $1, 5, \ldots, 25$ is $\overline{T}_{\text{standardised}} \approx 3.26 \pm \sigma$ with a small standard deviation $\sigma \approx 0.14$. This means that, empirically and on average, only slightly more than 3 solutions have been returned for each recursive call on a tail, and that this rate does not increase with the number of segments in the source route.

The empirical results for C and for $\overline{T}_{\text{standardised}}$, respectively, will be subject to further discussion in the next section.

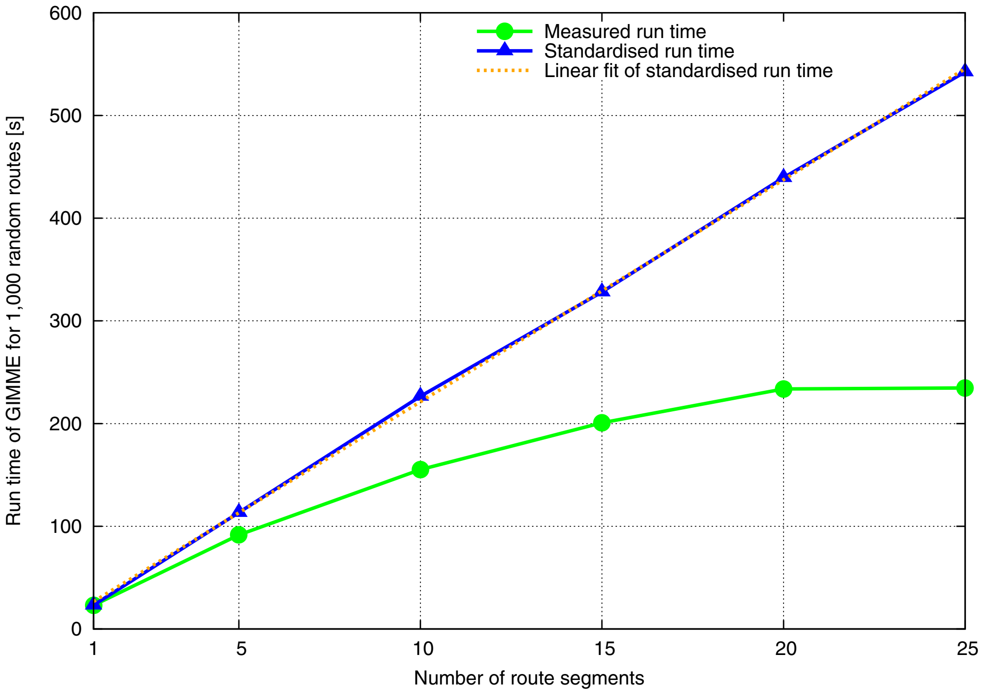


Fig. 9 Measured run time, standardised run time, and linear fit of the standardised run time of GIMME, applied to 1,000 line locations with various numbers of segments.

4.2.3 Amortised average run time

This section discusses the amortised average run time of GIMME in more formal terms, using the empirical results for C and $\overline{T}_{\text{standardised}}$ of the previous section, Section 4.2.2. The following more theoretical results have been confirmed by the experimental results of the previous section, and shed more light by explaining them.

Algorithm 1 first calls GEOMETRYMATCHES. This routine applies the GM algorithm ([26], see also Section 2.3.4) to all n edges of the source route e and constructs the respective candidate lists. Since GM does the work for every of the n source edges separately, i.e. each time working on an input of constant size, a linear (worst-case and average) run time should be expected. Further, also the percentage of the run time for GM on the total run time

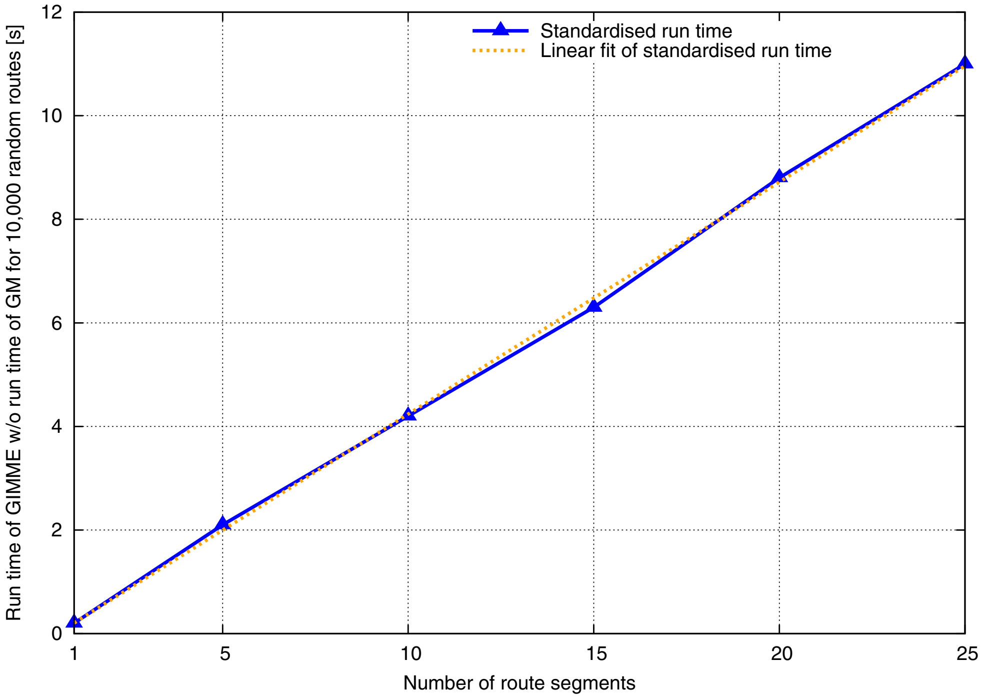


Fig. 10 Standardised run time, and linear fit of the standardised run time of GIMME without run time of GM, applied to 10,000 line locations with various numbers of segments

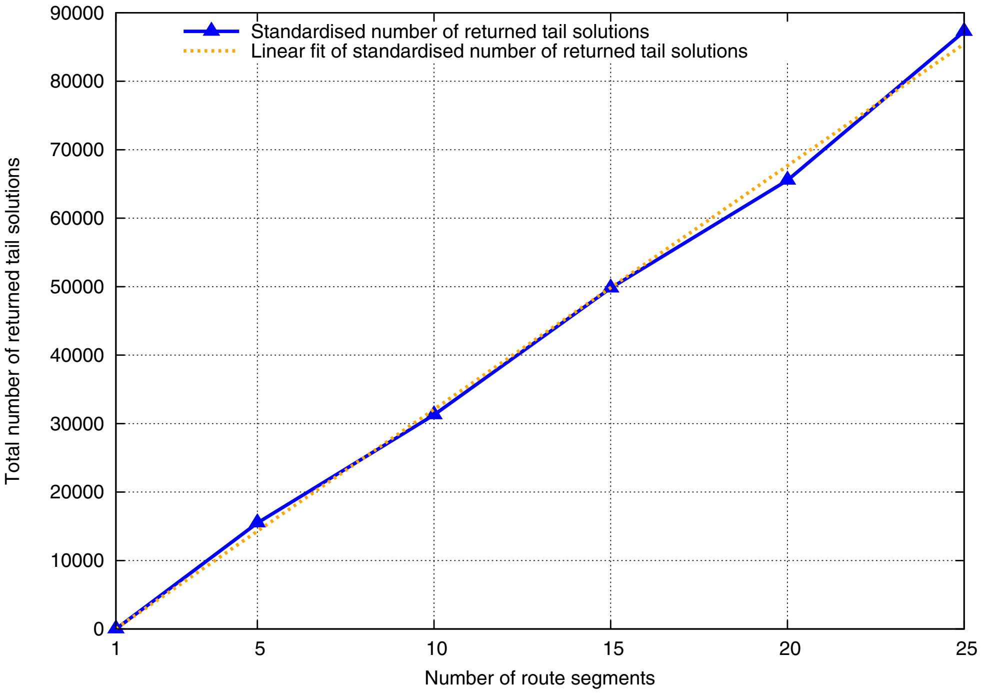


Fig. 11 Total number of returned solutions for the tail, standardised with respect to 1,000 successfully matched random line locations, with various numbers of segments, respectively

of GIMME is of interest. Both has been measured during experiments given in Section 4.2.2, confirming the theoretical expectations of a linear relation between the runtime for GEOMETRYMATCHES and the route length.

The second routine called by Algorithm 1 is Algorithm 2 (SOLUTIONS). This is the actual divide-and-conquer algorithm in question, and the remainder of the section will focus on analysing its average run time behaviour (see below). Moreover, Algorithm 1 calls LARGESTADMISSIBLESOLUTION and FINDFIRSTCLOSEDLINESOLUTION: their (worst-case and average) run time is linear in the length of their input, the set of solutions returned by the first recursive call in line 6 of Algorithm 1. By the empirical result from Section 4.2.2 for $\overline{T}_{\text{standardised}}$ we know that on average, the size of this set does not increase with n, the source route length. Rather the average size is asymptotically bounded by a small constant (slightly above 3), and thus the aforementioned two routines run in amortised constant average time.

In order to settle the run time behaviour of a divide-and-conquer algorithm like SOLUTIONS, one needs to solve the underlying recurrence. The terms of this recurrence are describing the computational cost for (i) dividing the input problem and for (ii) combining the partial solutions.

In brief, the cost of dividing the input e of Algorithm 2 (SOLUTIONS) into head and tail is that of list operations with constant run time. The cost for combining the partial solutions does not directly depend on n, the size of the input source route e. Instead this cost depends on (i) $|M|$, the number of candidates, because all sizes of lists and sets, and with it also the number of iterations in the respective loops of the program code, depend on this number; and on (ii) $|T|$, the number of solutions returned by a recursive call (the responsible code is found in the lines 26 - 34 of Algorithm 2).

More precisely: The cache operations in lines 5, 6, and 37 in Algorithm 2 are assumed to have amortised constant average run time. The computation of the power set of the set of candidates in line 12 of Algorithm 2 (SOLUTIONS) is done with a well-known recursive standard implementation for this task (no instructive pseudo code is given). It has the expected run time of $\Theta(2^{|M|}) = \Theta(2^C)$ where C is the empirical maximum length of a candidate list determined in Section 4.2.2. Further, GIMME has to consider a maximum of $\sum_{k=0}^{C} \frac{C!}{k!}$ permutations of subsets of a candidate list (see e.g. [3]). Both stated terms may appear large, but with our result $C = 3$ (see previous section) it is $2^C = 8$ and $\sum_{k=0}^{C} \frac{C!}{k!} = 16$.

Hence the total number of considered directed paths $d \in D$ (line 18 in Algorithm 2) for all subsets $S \in P$ (line 13 in Algorithm 2) is bounded by 16.

Algorithm 2 uses Algorithm 3 (CONNECTEDPERMUTATIONS) to construct the set of connected permutations for a set of edges. Subsequently, Algorithm 3 is called on each subset of the respective candidate set. It generates the connected permutations using the following recurrent relation:

Let ϵ be a set of edges. A connected permutation of ϵ, say π, is either (e) if $\epsilon = \{e\}$ (i.e. ϵ is singleton), or there must be $e \in \epsilon$ such that there is a connected permutation of $\epsilon \setminus \{e\}$, say ρ, such that the permutation resulting from prepending e to ρ equals π.

The (worst-case and average) run time for the call of Algorithm 3 in line 14 is constant since the length of its input is constantly bounded, as are the inputs of Algorithm 4 (LEFTEXTENSIONS) called within Algorithm 3 (line 15): $|S|$ is bounded by $C = 3$, a singleton with only one edge is passed to Algorithm 4 together with D, and $|D|$ is bounded by $(C - 1)! = 2! = 4$.

In more detail: the run time of Algorithm 4 is in $O(|e| \cdot |D|)$ where e is the first and D is the second parameter of LEFTEXTENSIONS. This is because in the worst case, $|e|$ elements in e are prepended to a copy of $d \in D$ in $|D|$ iterations of the loop (prepending a single element is done in constant time). Notice that the run time of creating the shallow copy of the list d in line 6 of Algorithm 4 is constant (i.e. essentially only an object maintaining the list is "cloned", holding a constant number of maintenance and reference fields, pointing to e.g. first and last list element, respectively). Regarding Algorithm 3, it is straightforward to see that the number of recursive calls to Algorithm 3 for a subset of maximal size 3 is bounded by 10. Since there are at most 8 subsets, the total number of calls to Algorithm 3 is bounded by $8 \cdot 10 = 80$ during the execution of one call of Algorithm 2 (SOLUTIONS).

Then, in lines 26 - 34 of Algorithm 2, GIMME checks every combination of a solution for the head for topological connectedness with a returned solution for the tail (the number of which typically remains small since only topologically connected paths are returned). The related work is done (i) in a loop for all solutions $s \in T$ (lines 27 - 31 in Algorithm 2), (ii) in the call to Algorithm 4 (LEFTEXTENSIONS) in line 32 in Algorithm 2), and (iii) in line 33 in Algorithm 2 which adds newly found solutions (i.e. lists of edges) as new elements to X, the tentative list of solutions (a list of edge lists).

Regarding (i), the run time of this loop is in $O(|T|)$ since in the worst case, during each of the $|T|$ iterations of the loop, one more element (a solution $s \in T$) is added to set X, an operation of constant run time.

Regarding (ii), by a previous consideration we already have that the run time of Algorithm 4 is in $O(|e| \cdot |D|)$ where e is the first and D is the second parameter of LEFTEXTENSIONS (see above). Therefore, the run time for (ii) is in $O(|d| \cdot |T|) = O(|T|)$ since d is a directed path constructed from at most $C = 3$ candidates for the head of the source route.

Regarding (iii), the operation of adding $|L|$ elements to the set X runs in $O(|T|)$ since $|L| \leq |T|$ elements are added in constant run time each.

By the empirical result from Section 4.2.2 for $\overline{T}_{\text{standardised}}$ we know that the average of $|T|$ is asymptotically bounded by a small constant, and thus the operations of (i), (ii), and (iii) all run in amortised constant average time.

Summarised, a relevant call to Algorithm 2 (SOLUTIONS, a recursive divide-and-conquer algorithm) has

(a) constant (worst case and average) time cost for dividing the problem, and
(b) amortised constant average time cost for combining the partial solutions.

The sum of both costs is an amortised constant average cost, which will be denoted by c_a.

By a result of Section 3.4, we have that n relevant recursive calls of Algorithm 2 are executed if a source route is matched successfully, or, as the corresponding recurrence,

$$\begin{aligned} T(n) &= 1 \cdot T(n-1) + c_a \\ &= n \cdot c_a \\ &= \Theta(n) \end{aligned}$$

That is, SOLUTIONS has an amortised average run time in $\Theta(n)$ for successful matches.

A successful match of GIMME has

(i) a worst-case and average time cost in $\Theta(n)$ for GEOMETRYMATCHES using GM, and
(ii) an amortised average time cost in $\Theta(n)$ for SOLUTIONS, and
(iii) amortised constant average time costs for either LARGESTADMISSIBLESOLUTION or FIRSTCLOSEDLINESOLUTION.

In total, GIMME has an amortised average time cost in $\Theta(n)$.

These more formal results are confirmed by the linear run time behaviour observed during the experiments described in Section 4.2.2. Note that for a failed match, the run time can only be smaller, and therefore in this case GIMME has a worst-case run time in $O(n)$.

4.2.4 Qualitative comparison of GIMME and the RNM-based approach

A majority of the recent road network matching (RNM) approaches are based on Buffer Growing (BG), cf. Section 2.3.2, Iterative Closest Point (ICP), cf. Section 2.3.1, or the combination and evolution of them [44]. The two most recent ones are the Delimited-Stroke-Oriented (DSO) algorithm [44] (see also Section 2.3.3), and NetMatcher [23].

Unfortunately, open source reference implementations are not available for the aforementioned RNM approaches. DSO as well as NetMatcher are quite complex algorithms, which makes a re-implementation very costly. Consequently, reference matching results allowing for a quantitative comparison to GIMME for the same pair of maps are very expensive to obtain.

Thus, this section gives a comparison of GIMME and the RNM-based approach on more qualitative terms. It is based on previously reported results and on the results of Section 4.2.1.

For a description of DSO, see Section 2.3.3. Like ICP, NetMatcher matches vertices first, and subsequently matches edges that are connected to matched vertices. Netmatcher assumes maps from the same vendor with different levels of detail, which is a strong limitation. Therefore, a qualitative comparison in terms of evaluation measures will focus on comparing GIMME to the RNM-based approach with DSO ($\mathrm{RNM}_{\mathrm{DSO}}$), since DSO (like GIMME) has no such restriction. On the other hand, run times for DSO are reported only for small map sections around 6,000 edges in Munich [44]. In contrast, the run times reported for NetMatcher were measured for road maps of a more practical size around 100,000 edges. Therefore, in Section 1 we have referred to the run times reported for NetMatcher.

In [44], evaluation measures are reported for the following experiment with DSO: sections of maps for Munich, Germany from TeleAtlas and NAVTEQ describing the road network in the downtown area have been matched. From all experiments described in [44], this is the one most similar to the first experiment described in Section 4.2.1: in both experiments, a TeleAtlas map for a road network in a German city is matched to the respective NAVTEQ map. On the other hand, neither the same pair of maps nor the same city or comparable map sizes have been used. Hence, the reported results only allow for a qualitative comparison.

During the aforementioned experiment, the success rate of DSO has been reported as 99.6% (termed ”matching correctness” in [44]). This rate reflects the probability of matching an edge of the road network successfully (i.e., correctly) with DSO. As before, we refer to $\mathrm{RNM}_{\mathrm{DSO}}$ as the RNM method establishing the required static mapping with DSO. The approach $\mathrm{RNM}_{\mathrm{DSO}}$ matches a linear route in the source map consisting of m edges to the target map by matching all of its edges individually to the target map, using the established static mapping, and by “glueing together” the corresponding edges or lists of edges to the final target route. The probability of a successful match can be estimated to $0,996^m$ since all individual edge matches must be successful for a successful match of the route. Notice that GIMME does *more* than merely glueing together matched edges: instead it keeps *multiple* matching candidates per edge, and then a “best” combination of candidates for all route edges is constructed. This helps to match e.g. long parallel roads with a short average distance to each other (a more detailed explanation follows below).

Moreover, for the aforementioned experiment with DSO, the following numbers of ”proper non-matches” (i.e., true negative matches), and of false negative matches have been reported in [44]: $n_{\mathrm{tn}} = 521$, $n_{\mathrm{fn}} = 122$. The respective error detection rate is $\frac{521}{521+122} \approx 0.810 = 81.0\%$. Consequently, for $\mathrm{RNM}_{\mathrm{DSO}}$ the probability of truthfully identifying a route as not having a homologous counterpart in the target map should be at least 81.0%: this is because a source route has no counterpart in the target map if at least one of its edges has no counterpart(s) in the target map, and because the probability of correctly classifying a first such edge is approximately 81.0%. Theoretically, if this first edge has been misclassified, i.e. if it rather has a match in the target map, the route could still be correctly identified as having no counterpart, if other edges without counterpart(s) exist in it. For some pairs of maps, this effect could cause an increase of the error detection rate.

These numbers now allow for a qualitative comparison with GIMME: during the experiments described in Section 4.2.2, matching routes consisting of different fixed numbers of edges with GIMME, also the resulting success rates and error detection rates have been determined by manual inspection. The number of positive matches (i.e. n_{p}) drops below 100 for routes with more than 25 segments: as has already been observed during the experiments described in Section 4.2.2, the longer a considered source route is, the more unlikely it is to find a correct match for it in the target map. Due to this drop, accuracy statistics for longer routes would not be reliable anymore, and thus longer routes have not been considered. Table 4 shows the observed success and error detection rates of GIMME, together with the estimated success and error detection rates for $\mathrm{RNM}_{\mathrm{DSO}}$. GIMME yields *higher* success rates than estimated for $\mathrm{RNM}_{\mathrm{DSO}}$, respectively. For a number of segments in the range of 1 to 20, the success rate of GIMME

Table 4 Success rates (q_{success}) and error detection rates ($q_{\mathrm{error\_detection}}$) for line locations with various fixed numbers of segments

Number of segments	GIMME		$\mathrm{RNM}_{\mathrm{DSO}}$ (estimated)	
	q_{success} [%]	$q_{\mathrm{error\_detection}}$ [%]	q_{success} [%]	$q_{\mathrm{error\_detection}}$ [%]
1	99.7	69.0	99.6	81.0
5	99.7	69.0	98.0	81.0
10	99.6	69.0	96.1	81.0
15	99.0	69.0	94.2	81.0
20	98.0	69.0	92.3	81.0
25	93.8	69.0	90.5	81.0

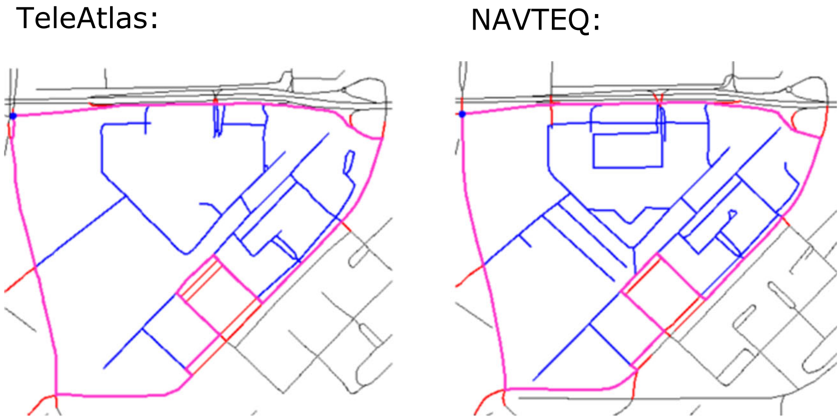


Fig. 12 Matching a circular route around "Stern-Center", Potsdam, Germany, passing "Nuthestraße", "Sternstraße", "Konrad-Wolf-Allee", and "Zum Kirchsteigfeld" (purple: route segments, blue: segments within the area surrounded by the closed line location, red: outer segments adjacent to the route segments)

degrades more slowly with the number of segments than the success rate of RNM_{DSO}: e.g. for routes with 20 segments, GIMME has a success rate of 98.0% , whereas the estimated success rate for RNM_{DSO} is only 92.3%, a difference of 5.7%. GIMME has a success rate of 93.8% for routes with 25 segments, whereas the estimated success rate for RNM_{DSO} is only 90.5%. The success rate of GIMME is better because the algorithm keeps multiple matching candidates per edge, and since a best combination of candidates for all route edges is constructed. That way, GIMME is capable of correcting a wrong choice subsequently, until a best admissible (i.a. topologically connected) path in the target map has been found.

For example, let us consider two parallel roads with a short average distance to each other, existing in both the source and in the target map. Let us assume that there is a significant offset between the two maps, and that one of the parallel roads needs to be matched. Because of the offset, the source road might on average be farther away from the homologous road than from the "wrong" second road, i.e. the one that is running parallel to it in close vicinity. Consequently, a road network matcher just matching edge-by-edge might make many wrong decisions here. In particular, the static mapping of an RNM-based approach stores only one matching candidate per edge, and consequently the approach is not capable of correcting such incorrect choices later. In contrast, GIMME stores *every* reasonable candidate edge and is therefore able to discard wrong decisions later. Provided that the wrong road deviates from the correct one at some point of the course, GIMME will detect it, discard the wrong decisions and finally construct the route from only the correct matches.

Notice that the error detection rate of GIMME did not increase with the number of segments per route. This means that, at least for the considered pair of maps, there is no increase in the error detection rate due to multiple unmatchable edges. Consequently, no such increase would be observed for the RNM-based approach either, and therefore the respective error detection rates for RNM_{DSO} in Table 4 are all estimated as 81.0%. GIMME showed a *lower* detection rate of 69.0%. Nonetheless it is noteworthy that also DSO's error detection rate is significantly lower than it's success rate like it was the case for both GIMME and OpenLR (see Section 4.2.1).

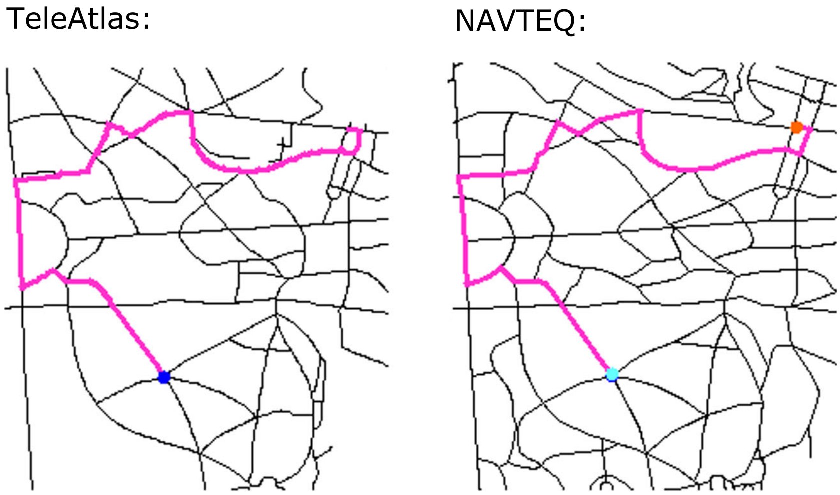


Fig. 13 Matching a route consisting of 25 road segments in Potsdam, Germany (driving direction from south to north, starting in "Brandenburger Vorstadt", passing "Neues Palais" in the west, proceeding further to the north via "Festungsweg", and ending in "Maulbeerallee")

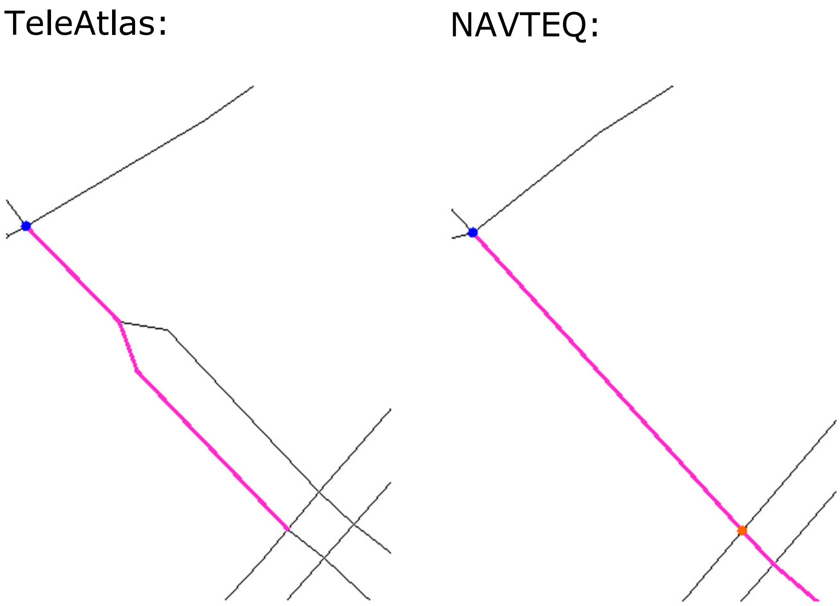


Fig. 14 Matching a bidirectional stretch of road on B273 (heading towards "Berliner Ring" in the northeast outskirts of Potsdam, Germany) with a physical barrier between the separated carriageways; *left*: representation with dual carriageways in the TeleAtlas map, *right*: representation as a bidirectional single carriageway in the NAVTEQ map

In practice, a good success rate is much more important than a good error detection rate, since the matching errors cause much more problems than the unmatched objects. This holds for both considered application domains, i.e. for RNM as well as for dynamic location referencing (DLR). For RNM, this holds because, "[. . .] the unmatched objects can be very easily marked as doubtful, whereas the process to detect the errors is time consuming and labour intensive, because erroneous matches need to be analysed one by one" [44]. For DLR, unmatched location references can easily be suppressed or replaced at the application level, e.g. a traffic-monitoring application can always drop an unmatched reference to a congested stretch of road, or perhaps replace it by a less accurate, more general textual warning. On the contrary, actually delivering a piece of wrong information about a jam (because of an unnoticed matching error) can irritate the users.

To summarise: For the limited scope of the experiments described in [44] and in Section 4.2.1, the RNM-based approach with DSO ("RNM_{DSO}") can be compared to GIMME in qualitative terms as follows: GIMME is expected to have a higher success rate than RNM_{DSO}. For all considered algorithms (RNM_{DSO}, GIMME, and OpenLR), the error detection rate is significantly lower than the success rate. The error detection rate of GIMME is expected to be lower than that of RNM_{DSO}. It is of note that the success rate is the more important measure for both application domains, i.e. for both RNM and DLR.

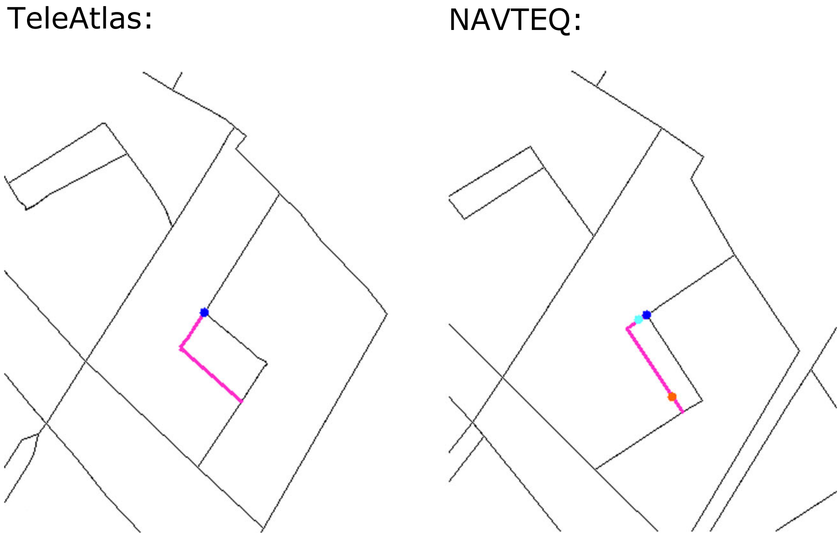


Fig. 15 Matching a short stretch of road consisting of only one edge in "Max-Wundel-Straße", Potsdam, Germany; *left:* source edge, *right:* candidate edge, trimmed by positive and negative offsets

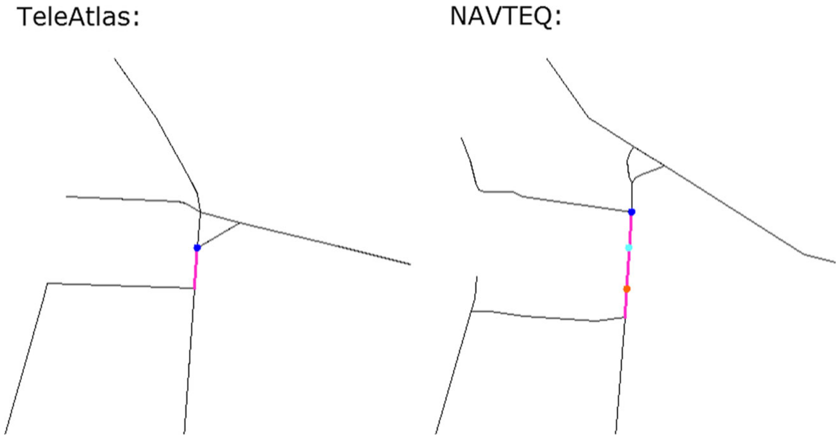


Fig. 16 A section of the two maps of Potsdam, Germany (around "Galliner Damm" / "Golmer Damm"), where it is difficult to decide for a human expert whether the found match should be considered as correct or not

4.2.5 Example matches

This section gives depicted examples for successful matches of circular and linear routes as resulting from GIMME, and also discusses a situation where GIMME fails to find the correct match, and how to remedy it.

Figure 12 shows a circular route of 3,133 meters length around "Stern-Center", Potsdam, Germany, passing "Nuthestraße", "Sternstraße", "Konrad-Wolf-Allee", and "Zum Kirchsteigfeld" (left: source route in the TeleAtlas map, right: matched route in the NAVTEQ map). The route segments are coloured in purple, the segments within the area surrounded by the closed line location are coloured in blue, and the outer segments adjacent to the route segments are coloured in red. The blue dot depicts the origin or start of the circular route, and the driving direction is clockwise.

Figure 13 shows a route consisting of 25 road segments in Potsdam, Germany, with a length of 1,936 meters. The driving direction is from south to north, starting in "Brandenburger Vorstadt" (again marked by a blue dot), passing "Neues Palais" in the West, proceeding further to the north via "Festungsweg", and ending in "Maulbeerallee").

Figure 14 depicts an example demonstrating that GIMME is capable of dealing with the situation that roads with a physical barrier between two separated carriageways might be represented as a bidirectional road with a single carriageway in one map, and as a road with dual carriageways in another: A bidirectional stretch of road on B273 (heading towards "Berliner Ring" in the northeast outskirts of Potsdam, Germany) is represented with dual carriageways in the TeleAtlas map, and as a bidirectional single carriageway in the NAVTEQ map. GIMME finds the correct match since GM accounts for such differences in representation by tolerating the resulting larger angular differences between homologous roads ([26], see also Section 3.3).

Figure 15 depicts a situation where GIMME *fails* to match a short stretch of road in "Max-Wundel-Straße", Potsdam, Germany, consisting of only one edge in the TeleAtlas map with a length of 81 meters. The GM algorithm finds a candidate edge of 82 meters length, and calculates a positive as well as a negative offset by projecting the source edge onto the candidate edge (depicted to the right of Fig. 15; the blue dot again marks the origin of the short stretch, i.e. the driving direction is from north to south; the light blue dot marks the position of the positive offset, and the red dot marks the position of the negative offset). Due to a significant angular difference between source and candidate edge, both offsets have relatively large absolute values (positive offset: 6 m, negative offset: 12 m). Thus, the remaining trimmed candidate route has a length of only 64 meters. Since the source route is very short, this is already below the percentage threshold for an admissible length of a candidate route which is $0.8 \cdot 81 = 64.8$ meters (see the definition of an admissible candidate route in Section 3.1).

Consequently, GIMME does not find a match in this case though the found candidate edge is obviously homologous to the source edge. This could be remedied by an increase of the respective threshold, but another possible remedy is to add an optional "snap mode" to GIMME. This mode is enabled at the application level (i.e., by an application making use of GIMME), and it is assumed that the application always passes source routes starting and ending at a graph node of degree greater than 2 (that is, starting at an intersection rather than somewhere between two intersections). Candidate routes are then either trimmed or expanded by "snapping" start and end position of the route to the nearest graph node of degree greater than 2 in the target map. The rationale is that intersections are usually prominent enough to occur in both maps. Therefore, the snapping strategy should identify the correct start and end position of the candidate route in most cases. It goes without saying that GIMME with "snap mode" succeeds to find the correct match in the described example.

It is of note that the experiments reported in Section 4.2 have been conducted *without* applying the outlined "snap mode", and that situations similar to the described one did not occur frequently.

Finally, Fig. 16 shows a section of the two maps of Potsdam, Germany, where it is difficult to decide for a human expert whether the found match should be considered as correct or not. Due to the large map differences, it is impossible to place a match in the NAVTEQ map in such a way that all topological relations to the surrounding map elements have exact correspondences in the TeleAtlas map. Consequently, an expert has essentially the choice to accept no matches at all, or to accept every match, or to define criteria for the "best of a bad bunch".

For the experiments conducted in Section 4.2.4, cases like that had been excluded and replaced by new randomly generated instances until no problematic locations were contained in the sample anymore.

5 Conclusion

A new approach to match linear or circular routes between two dissimilar maps has been presented. It is highly accurate and map-independent, but access to both involved maps is required. This requirement is met whenever an omniscient matching centre with access to both the source and the target map can be used. If location references need to be transmitted from a sender to a receiver, the algorithm computes the matches at the receiving side, using both maps. In such a setting, the approach can also be combined with a bandwidth-efficient dynamic location referencing method like e.g. OpenLR to obtain a compact format before the descriptive data is transmitted. Notice that in this case run time will increase due to the overhead of calling the additional bandwidth-efficient method for every location.

The new approach exploits the fact that information from both maps is available. It achieves high accuracy by comparing geometrical route descriptions derived from both maps. It advances on the path of previous methods in the area of road network matching and map conflation like e.g. Geometry Matching and Buffer Growing. In an experimental evaluation, the new approach has been compared to TomTom's OpenLR, and the results clearly demonstrate the increased accuracy of the resulting matches: it is currently capable of mapping closed line locations (i.e. circular routes) from a TeleAtlas map to a NAVTEQ map on-the-fly with a success rate of 97.5% (OpenLR: only 82.5%), and also capable of mapping short line locations (i.e. linear routes) on-the-fly between the same maps, with a success rate of 99.7% (OpenLR: 91.9%).

References

1. Besl PJ, McKay ND (1992) A method for registration of 3-D shapes. IEEE Trans Pattern Anal Mach Intell 14(2):239–256
2. Bofinger JM (2001) Analyse und Implementierung eines Verfahrens zur Referenzierung geographischer Objekte. http://www.ifp.uni-stuttgart.de/lehre/diplomarbeiten/Bofinger/Diplomarbeit. Online; Diplomarbeit, German
3. Comtet L (1974) Advanced Combinatorics, chap. 1, p. 75. Reidel, Dordrecht. Problem 9
4. mobile.info Consortium (2007) mobile.info Final Project Report. http://www.mobile-info.org/prom/mobileinfo.nsf/DocID/D6D289BDF68A0584C125739B00547F9E/$file/Mobile_Info_Schlussbericht.pdf. [Online; accessed 26-November-2014]
5. Creative Commons (CC). creative commons Attribution-NoDerivs 3.0 Unported. http://creativecommons.org/licenses/by-nd/3.0/legalcode. Online; accessed 26-November-2014
6. Demir C (2002) A new location referencing method for unique reconstruction of an object on a second map (ROSA). In: Proceedings World Congress on Intelligent Transport Systems 2002. Chicago, Illinois, USA
7. Dorenbeck C (2000) Method for generating a location reference instance within a digital map. http://www.google.com/patents/EP1020832A1?cl=en. EP Patent App. EP19,980,107,675
8. Duckeck R, Hendriks T, Heifling M, Otto HU, Pfeiffer HW, Wevers K (2003) Specification of the AGORA location referencing method Version 1.0. Information Society Technologies (IST), AGORA Project
9. Duckeck R et al (1997) Rules for defining and referencing an Intersection Location (ILOC); Detailed Location Referencing (DLR) for ITS based on ILOCs. ERTICO Committee on Location Referencing, Report, Version 1.0
10. EasyWay Consortium. DATEX II website. http://www.datex2.eu (2009–2014). [Online; accessed 26-November-2014]
11. EasyWay Consortium (2014) OpenLR extension 1.5 for DATEX II. http://www.datex2.eu/content/openlr-extension-15-0. [Online; accessed 26-November-2014]
12. Free Software Foundation Europe. Open Standards. http://fsfe.org/activities/os/os.en.html. Online; accessed 26-November-2014
13. Goodrich M, Tamassia R (2002) Algorithm design: Foundations, Analysis, and Internet Examples. Wiley, New York
14. Gupta P, Agarwal V, Varshney M (2012) Design and analysis of algorithms. PHI learning
15. Hahlweg C et al (2000) GOODLANE - An approach to location referencing for telematic applications. In: Proceedings World Congress on Intelligent Transport Systems 2000. Torino, Italy
16. Hendriks A (2002) A method and system for referencing locations in transport telematics. http://www.google.com/patents/EP1225552A1?cl=en. EP Patent App. EP20,010,101,123
17. Hendriks T, Wevers K (2004) AGORA-C location referencing - Specification, applicability and testing results. In: Proceedings World Congress on Intelligent Transport Systems. Nagoya, Japan
18. Hiestermann V (2008) Map-independent location matching certified by the AGORA-c standard. Transportation Res Part C: Emerg Technol 16:307–319
19. Hummelsheim K et al (2003) Location referencing change request for ISO. AGORA Project Consortium, Deliverable 6.4, Version 1.0
20. International Organization for Standardization (ISO). https://www.iso.org/obp/ui/#iso:std:iso:ts:21219:-2:ed-1:v1:en. Online; accessed 26-November-2014
21. International Organization for Standardization (ISO) ISO 17572-3 Intelligent Transport System (ITS) Location Referencing

for Geographic Database Part 3: Dynamic Location References. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=45962. Online; accessed 26-November-2014

22. International Organization for Standardization (ISO). ISO standards catalogue. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=37500. Online; accessed 26-November-2014
23. Mustière S, Devogele T (2008) Matching networks with different levels of detail. GeoInformatica 12(4):435–453
24. Pandazis JC (1999) Extensive Validation of IDENtification Concepts in Europe. EVIDENCE Consortium, Final Report v2.3.3
25. ROSATTE consortium. ROSATTE website. http://tn-its.eu/rosatte-project. Online; accessed 26-November-2014
26. Sämann R (2014) Bestimmung einer Bewertungsmetrik zum Vergleich digitaler Straßennetze für Verkehrsflusssimulation und Routing von Einsatzkräften. Master's thesis, Institut für Bauinformatik Leibniz Universität Hannover in cooperation with Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Hannover, Germany
27. Schneebauer C, Wartenberg M (2007) On-The-Fly Location referencing — methods for establishing traffic information services. IEEE Aerosp Electron Syst Mag 22(2):14–21
28. Svensk P, Wikström L (2012) ROSATTE — TRIONA Georeferencing Methods. http://tn-its.eu/docs/emaps/eMaPS-D2.42-Triona-Georeferencing-methods-v12-130321.pdf. Online; accessed 26-November-2014
29. TomTom International B.V. http://openlr.org/data/docs/OpenLR-Introduction.pdf. Slides 42–43; Online; accessed 26-November-2014
30. TomTom International B.V. OpenLR™Users. http://www.openlr.org/users.html. [Online; accessed 26-November-2014]
31. TomTom International B.V. OpenLR™website. http://www.openlr.org (2009–2014). [Online; accessed 26-November-2014]
32. (2012) TomTom International B.V., German Aerospace Center (DLR): OpenLR™ White Paper Version 1.5. http://openlr.org/data/docs/OpenLR-Whitepaper_v1.5.pdf. [Online; accessed 26-November-2014]
33. Transport Protocol Experts Group: TPEG specifications Part 6: Location Referencing for Applications
34. Transport Protocol Experts Group (2012) Intelligent Transport Systems (ITS) Traffic and Travel Information (TTI) via Transport Protocol Experts Group, Generation 2 (TPEG2) - Part 22: OpenLR location reference (TPEG2-OLR_1.0/002) http://www.tisa.org/tisa-products/tisa-documents/
35. (2011) Traveller Information Service Association (TISA): TISA website. http://www.tisa.org/. [Online; accessed 26-November-2014]
36. T'Siobbel S, Landwehr M, Mahiou R et al (2011) ROSATTE — Deliverable D4.1: Description of applicable and viable data integration methods. http://tn-its.eu/docs/rosatte/ROSATTE-D41-Data-integration-methods-v13-final.pdf. Online; accessed 26 November 2014
37. Via Licensing Corp. (2012) AGORA-C patent submission. http://www.vialicensing.com/licensing/agorac-patentcall.aspx. Online; accessed 26-November-2014
38. Volz S (2006) An iterative approach for matching multiple representations of street data. In: Hampe M, Sester M, Harrie L (eds) ISPRS Vol. XXXVI., ISPRS workshop - multiple representation and interoperability of spatial data. Hannover, Germany
39. Walter V (1997) Zuordnung von raumbezogenen Daten - am Beispiel der Datenmodelle ATKIS und GDF. Ph.D. thesis, Deutsche geodätische Kommission (DGK) Reihe C, Nummer 480
40. Walter V, Fritsch D (1999) Matching spatial data sets: a statistical approach. Int J Geogr Inf Sci 13(5):445–473
41. Wartenberg M (2006) Algorithms for Location Referencing. Jahresbericht der Deutschen Mathematiker-Vereinigung (DMV) 01/2006
42. Wevers K, Hendriks T (2005) AGORA-C On-the-Fly Location Referencing
43. Wevers K, Hendriks T (2006) AGORA-C Map-Based location referencing. J Transp Res Board 1972(1):115–122
44. Zhang M (2009) Methods and Implementations of Road-Network Matching. Ph.D. thesis, Leibniz Universität Hannover, Hannover, Germany
45. Zhang M, Meng L (2007) An iterative road-matching approach for the integration of postal data. Comput Environ Urban Syst 31(5):598–616