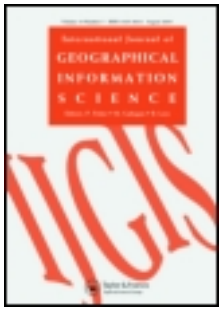

International Journal of Geographical Information Science

Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tgis20

Ad hoc matching of vectorial road networks

Eliyahu Safra [a], Yaron Kanza [b], Yehoshua Sagiv [c] & Yerach Doytsher [a]

[a] Mapping and Geo-Information Engineering, Technion, Haifa, Israel

[b] Computer Science, Technion, Haifa, Israel

[c] School of Engineering and Computer Science, The Hebrew University, Jerusalem, Israel

Available online: 18 Apr 2012

To cite this article: Eliyahu Safra, Yaron Kanza, Yehoshua Sagiv & Yerach Doytsher (2012): Ad hoc matching of vectorial road networks, International Journal of Geographical Information Science, DOI:10.1080/13658816.2012.667104

To link to this article: http://dx.doi.org/10.1080/13658816.2012.667104

International Journal of Geographical Information Science
i*First*, 2012, 1–40

Taylor & Francis
Taylor & Francis Group

Ad hoc matching of vectorial road networks

Eliyahu Safra[a]*, Yaron Kanza[b], Yehoshua Sagiv[c] and Yerach Doytsher[a]

[a]*Mapping and Geo-Information Engineering, Technion, Haifa, Israel;* [b]*Computer Science, Technion, Haifa, Israel;* [c]*School of Engineering and Computer Science, The Hebrew University, Jerusalem, Israel*

(*Received 13 January 2010; in final version received 24 January 2012*)

In integration of road maps modeled as road vector data, the main task is matching pairs of objects that represent, in different maps, the same segment of a real-world road. In an *ad hoc integration*, the matching is done for a specific need and, thus, is performed in real time, where only a limited preprocessing is possible. Usually, *ad hoc* integration is performed as part of some interaction with a user and, hence, the matching algorithm is required to complete its task in time that is short enough for human users to provide feedback to the application, that is, in no more than a few seconds. Such interaction is typical of services on the World Wide Web and to applications in car-navigation systems or in handheld devices.

Several algorithms were proposed in the past for matching road vector data; however, these algorithms are not efficient enough for *ad hoc* integration. This article presents algorithms for *ad hoc* integration of maps in which roads are represented as polylines. The main novelty of these algorithms is in using only the locations of the endpoints of the polylines rather than trying to match whole lines. The efficiency of the algorithms is shown both analytically and experimentally. In particular, these algorithms do not require the existence of a spatial index, and they are more efficient than an alternative approach based on using a grid index. Extensive experiments using various maps of three different cities show that our approach to matching road networks is efficient and accurate (i.e., it provides high recall and precision).

General Terms: Algorithms, Experimentation

Keywords: road map; spatial network; integration; matching polylines; location-based join; corresponding objects; conflation

1. Introduction

Vectorial road networks are digital representations of road maps. They can be used in applications such as finding the shortest route between two given locations, providing an estimation of the time it takes to get from one location to another, and so on. Such applications may need to use both the spatial and nonspatial properties of roads. Integration of two road maps makes it possible for the applications to associate each road with all of its properties, even those that are represented in only one map. For example, consider two road maps of some city. Suppose that only the first road map includes buildings in the city with the roads leading to them while only the second road map includes, for each segment

*Corresponding author. Email: esafra@iec.co.il

ISSN 1365-8816 print/ISSN 1362-3087 online

http://dx.doi.org/10.1080/13658816.2012.667104
http://www.tandfonline.com

of a road, the direction of the traffic and the speed limit in this segment. An integration is needed for estimating the minimal time it takes to get from a certain building in the city to another building.

In many scenarios, it is required to integrate road maps for a specific user need. We refer to such cases as *ad hoc integration*. In an *ad hoc* integration, the matching is done in real time and with almost no preprocessing. A typical scenario of *ad hoc* integration is when a small fragment of a road network should be matched with a larger network. For example, consider a case where a user marks a start and an end location on some road map in order to compute the route between them. Suppose there is also a second map with information that can improve the computation of the route (e.g., data about the traffic condition in each segment). In an *ad hoc* integration, the relevant fragment of the first map will be extracted and matched with the second map prior to the computation of the route. A second example is when a route is already computed over one map and the segments of this route should be matched with another road network for retrieving information about them.

In an *ad hoc* integration (and in other integration scenarios), the efficiency of the matching process is crucial. In particular, efficiency is required in applications on the World Wide Web for two main reasons. First, Web applications are usually interactive and, hence, answers to user requests should be provided immediately (within a few seconds at most). Second, applications on the Web are required to handle many users concurrently.

Another case where the efficiency of the process is vital is of applications in car-navigation systems. Such applications should complete their task instantly in order to provide relevant results while driving. Furthermore, the applications may need to run in devices with a limited processing power.

Several methods for integrating road maps were proposed in the past (see Walter and Fritsch 1999, Gabay and Doytsher 2000, Doytsher and Filin 2000). However, these methods are designed for finding answers that are as accurate as possible without taking efficiency into consideration. Since these methods require a long computation time, they are not suitable for scenarios where efficiency is crucial.

In this article, we consider maps in which roads are represented by polygonal lines (we use the two terms *polyline* and *line* as an abbreviation of polygonal line). The road vector data are represented in the *Network Model*. In this model, whenever two polylines intersect, the intersection is represented by a node. For comparison, in the *Spaghetti Model,* road intersections are not always represented by a node. For a discussion about the differences between the two models, see Scholl *et al.* (2002).

An integration of two such maps is essentially a matching between pairs of polylines that represent the same road in the two maps. The novelty of our approach is in matching roads based mainly on locations of endpoints of polylines rather than trying to match lines according to some distance measure that takes into account all the points on the lines (e.g., using the Hausdorff distance, or associating two lines if all the points of one line are contained in a buffer around the other line). There are two important advantages to our approach. First, integration can be done efficiently by limiting the number of complex geometric calculations that need to be done. Second, we want our techniques to be general in the sense that they would not require the existence of any particular property of roads, other than endpoint locations. Different from other properties, locations always exist for objects in spatial databases. Also, locations have the same semantics in different maps, so we can compare them without worrying that we will end up comparing unrelated properties. In particular, our approach matches polylines based on their local environment without using (in each matching of two polylines) complex topological properties whose computation is expensive, for example, properties of the entire network. That is, on the

one hand, our algorithms match (in each step) pairs of polylines merely according to the matching of the endpoints, rather than matching sets of polylines or subnetworks. On the other hand, the matching of two polylines is affected by the environment of each polyline, because the matching of the endpoints is done while considering for each endpoint its neighboring endpoints. This is important because using the topology, for example, by applying transformation such as shifting the axes, rotation, or scale (stretch) till finding the best match between the networks, may increase the complexity of the computation, since it requires expensive algebraic or geometrical computations. Furthermore, relying strictly on the topology can be problematic when information is incomplete. For example, we may need to match an intersection of three roads in one map with an intersection of four roads in a second map, due to the fact that some roads are represented in only one of the maps. In our approach, in comparison, two endpoints of polylines can be matched if one is near the other, relative to the other endpoints in their environments, even if they are intersections of different numbers of roads.

It may seem an easy task to match roads using their endpoint locations. However, this is not the case for the following reasons. First, locations are not accurate, so usually two maps represent the same real-world entity in two different locations. Second, endpoints may be chosen differently in the two maps and, hence, the location of an endpoint in one map may be inside a polyline in the other map. Furthermore, when a road is represented as a polyline rather than as a curve, the representation is just an approximation of the real-world line, and the two maps can use different approximations. Third, information might be incomplete so that a road or a segment of a road may appear in only one map.

In this article, we introduce a method for integrating two vectorial road networks, whereby polylines are matched according to the locations of their endpoints. Essentially, the method is a two-stage process. Initially it finds the endpoints of the polylines in the two sources and computes a partial matching between them. Then, it finds a partial matching between the polylines, according to the matching of the endpoints.

Endpoints should be matched based on proximity, to deal with inaccurate locations, while taking into account that not every two adjacent endpoints should be matched, for example, we may not match points a and b that are near each other if there is a point c such that the distance between a and c is smaller than the distance between a and b. To do so, we present two ways of matching endpoints, namely, the *semantics* and the *semantics*. Under the semantics, two endpoints are matched if each one is the nearest neighbors of the other. Under the semantics, two endpoints are matched if at least one of the points is the nearest neighbor of the other.

According to the matching of the endpoints, we match the polylines. This is not a simple task, because two polylines may represent overlapping segments of a real-world road while their endpoints are far from each other. That is, an endpoint of one polyline may reside between the endpoints of a corresponding polyline. Hence, matching only pairs of polylines whose endpoints are all matched is insufficient for finding all the polylines that represent the same road segments in the real world. In other words, there may be pairs of polylines whose endpoints are not near each other, and yet they should be discovered by the algorithm.

We present three methods that resolve such cases. One method uses the matched endpoints and segments to discover such corresponding roads. We start by matching polylines according to endpoints, and then we check, recursively, polylines that are connected to previously matched polylines. The other two methods use a spatial index to retrieve, for each polyline, candidate matching polylines. One index-based method builds buffers around the endpoints when computing the matching, and the other method builds buffers around the

whole polylines. For each polyline p of one source, the algorithm retrieves the polylines of the other source that intersect the buffer of p and then checks for each retrieved polyline if it is close enough to p. We show that building buffers around endpoints is more efficient than building buffers around entire polylines and that the first method – the one which is not based on using an index – is more efficient than the index-based methods.

The above explanation illustrates how our algorithms deal with the difficulties of endpoint-based matching of road vector data. For tackling the problem of inaccurate locations, the algorithms use approximations, yet they do that while taking into account the effect of neighbors on the matching of nodes, so that the matching will be accurate for either dense or sparse networks. The test whether two nodes should be matched is local, and thus, is efficient. In order to handle correctly the cases where endpoints are chosen differently in the two maps and information is incomplete, a partial matching is computed (instead of a complete matching) and then it is used to traverse the network and match polylines even when the polylines overlap but their endpoints do not match.

In order to show the efficiency and the effectiveness of our techniques, we conducted extensive experimentation on real-world datasets from several different sources. We tested our algorithms on various scenarios that differ in the topography of the test area, the accuracy level of the input datasets, and the number of objects within each input datasets. In the tests, we compared the and the semantics. Also, we studied the effect of computing the matching using only endpoints that satisfy a given condition about the number of roads that intersect them. Our tests show that the proposed integration methods are efficient and accurate, that is, they provide high recall and precision. The tests also show that the best performance in terms of efficiency is achieved when using the semantics while considering only endpoints that are intersections of three or more roads. In terms of correctness, the quality of the result of an algorithm depends on features of the input datasets, such as the connectivity of the network and the density of the dataset. These features and their effect are later discussed in depth.

The outline of this article is as follows: Section 2 discusses related work. Section 3 introduces the framework and formally define the problem. Section 4 describes our algorithms for integration of two road maps. Section 5 explains how to use spatial indexes for integration of road maps. Section 6 defines how the quality of a matching algorithm can be measured. Section 7 presents our extensive tests and their results are. Finally, Section 8 concludes.

2. Related work

Integration of geospatial information has been the focus of several papers (see Noronha 2000, Laurini *et al.* 2002). A general approach for data integration, based on an architecture of wrappers and mediators, has been proposed by Wiederhold (1992, 1999). In such an architecture, data are collected from different sources by specialized wrappers, and they are integrated in a mediator. This architecture has been adapted to geospatial vector data in the work of Boucelma *et al.* (2002). In this work, Web Feature Servers provide vector data in the form of XML. A mediator translates a request into a set of XQuery queries—one for each source, and the answers to these queries are combined to provide the result. Parent *et al.* (2006) have developed a system to support the integration of geospatial data from heterogeneous sources where the representation may have semantic and cartographic flexibility, for example, having multiple resolutions in the sources. They, as well, transform geographic data to an XML representation, using wrappers, and apply queries over this representation. Although it is important to notice that in these papers, the main

task is providing a general framework for integration rather than specifying how to match geographic features.

Several attempts were made to integrate geospatial data using an ontology, in ontology-driven geographic information systems (Fonseca and Egenhofer 1999, Uitermark *et al.* 1999, Fonseca *et al.* 2002). In such systems, geographic features of vector datasets are represented as objects. These papers studied the use of concepts, names, and taxonomies to find corresponding objects. In many cases, however, an ontology (and schema information in general) is not sufficient for accurately finding all the corresponding objects, especially when the data sources are heterogeneous and the information is incomplete.

The papers mentioned above deal with integration of geospatial data in general. Several other papers specifically investigated matching lines and integrating road vector data. Some papers proposed algorithms that mainly use geometric attributes, whereas in others, the proposed algorithms match objects according to nongeometric attributes, such as types and names of the corresponding geographic entities.

Rosen and Saalfeld (1985) and Saalfeld (1988) suggested a two-step iterative process for matching polylines. In the first step, several objects are matched using their location (based on proximity) and some other attributes. In the second step, a rubber-sheet transformation is applied in order to correct the locations of objects that were not matched in the first step. These objects are then matched according to the corrected locations. A method that uses triangulation and rubber-sheet transformation, but for conflating vector data with raster data (i.e., vector to imagery conflation), has been suggested by Chen *et al.* (2006).

Gabay and Doytsher (2000) proposed an integration algorithm, for road vector data, that uses buffers. All the polylines of one source are augmented with a buffer. When a polyline from the other source is completely contained in the buffer of some polyline, the two polylines are considered a matching pair. Another matching algorithm, for vector data, that uses buffering has been suggested by Ware and Jones (1998). An approach similar to using buffers employs the Hausdorff distance, of Hangouet (1995), as the appropriate metric for measuring the distance between lines (e.g., see Volz and Walter 2004).

Filin and Doytsher (1999; and see also Doytsher and Filin 2000, Doytsher *et al.* 2001) introduced a method that initially matches road intersections using a point-matching algorithm. Then, it uses the topology of polylines to propagate the matching to the lines. Finally, a relative-compatibility measure, based on the average distance between lines, is computed for each pair of matched lines. Utilizing topological relationships between polylines for map conflation, in integration of road vector data, has also been proposed by Haunert (2005).

The above line-matching algorithms are not efficient in the sense that they all use complex geometric computations. Furthermore, these matching techniques use in their computations all the points that are contained in the polylines, rather than using merely the endpoints of the polylines.

Cobb *et al.* (1998) proposed a method, for conflation of vector data, that uses both geometric attributes, such as location and shape, and nongeometric attributes such as type and name. Initially, the method finds associations between similar attributes from the different sources. These associations are used to construct a set of rules that specify how to match objects. Each rule is given a significance rank. The geographical objects are matched by applying the rules in the order of their significance rank.

Dealing with sources of vector data at different scales was studied by Devogele *et al.* (1998). In their work, sets of features of one source that are contained in a buffer around a feature in the other source are matched. Mustière and Devogele (2008) studied integration

of networks that have different levels of detail by using Hausdorff distance between polylines and using topological properties of the networks that are not affected by the scale. Their focus, however, was on the data model and the general process for such tasks but not on how to execute the computation efficiently.

Matching road vector data at different scales was also studied by Sester *et al.* (1998) and by Walter and Fritsch (1999). Their approach was to solve a maximization problem over the space of all the possible matchings. To that end, they defined a measure called *mutual information* that evaluates a matching of two datasets based on several parameters for each pair of matched objects (overlap length, shape similarity, form, and existence of a connection). The goal is to find the matching that provides the maximal mutual information, and this is done using the A* search algorithm. Walter and Fritsch (1999) use buffers, topological properties (e.g., node degree), and geometrical properties (e.g., angle, length, and position) for calculating the mutual information of matchings. The approach is based on statistical investigations of manually matched training datasets. They represent all the possible matchings as a search tree and apply A* search over it. A manual postprocessing step completes the matching of lines that were not matched automatically. For dealing with different scales, Sester *et al.* (1998) aggregate objects that are matched to a single object in the other source. They use machine-learning techniques to do so. Initially the algorithm is trained over a small example and learns how to classify the objects and aggregate them accordingly. It generates mapping rules by finding regularities in the given example. Then, an association tree that represents all the possible matchings is constructed. The best match in the tree is discovered in the A* search. Note that the training phase of the learning algorithm requires that initially a matching of some small test case will be done manually by a human. Li and Goodchild (2010) presented a method of matching objects in geospatial datasets using linear programing. They showed how their approach can be applied to matching street networks; however, the approach requires a preprocessing step to handle issues caused by differences in the generalization of distinct streets, and it works well only when the result matching is a bisection. It does not deal gracefully with incomplete information or with scale differences.

The methods of Cobb *et al.* (1998), Sester *et al.* (1998), and Walter and Fritsch (1999) all required prior knowledge about the datasets; moreover, Sester *et al.* (1998) and Walter and Fritsch (1999) also included a preprocessing step that requires human intervention – generating a matching example for a learning algorithm. The semiautomatic method of Xiong and Sperling (2004) required a postprocessing step where the user corrected an initial matching. Thus, these methods are not suitable for *ad hoc* integration or for applications that require an immediate response.

There are also several commercial tools for conflation of spatial data in general (e.g., ArcGIS, ERDAS IMAGINE, INTERGRAPH, FEATURE ANALYST) and of vector data in particular (e.g., MapMerger). However, these tools are not designed for *ad hoc* integration.

3. Framework

In this section, we present our framework. We formally define the notion of a road map in a geospatial database, and we present the notion of correspondence between roads when matching two road networks. We discuss the notation of a matching algorithm that finds corresponding road segments, and we describe the result of such an algorithm. We also explain how the error in the location of polylines affects the matching algorithms.

3.1. Road vector data

A *road map*, in the form of *road vector data*, represents a network of real-world roads, using nodes and edges. The *nodes* (also called *topological nodes*) are either *intersections*, where two or more roads meet, or *road ends* where roads terminate without intersecting other roads. The *edges* are *road objects*. Note that under this interpretation, a road may start or end at an intersection but never includes an intersection as an intermediate point.

A road object is represented by a *polyline*. A polyline is a continuous line composed of one or more line segments, such that every two consecutive segments intersect only in their common endpoint while nonconsecutive segments do not intersect. In some places, where it is clear from the context, we use the term *line* for a polyline. Formally, a polyline l is a sequence of points $p_1, \ldots, p_n$. Every two successive points p_i and p_{i+1}, in the sequence, define a *segment* of the polyline. The points p_1 and p_n are the *endpoints* of l.

As noted earlier, the endpoints are the nodes of the road map. The *degree* of a node p is the number of polylines that have p as one of their endpoints.

A road map is a geospatial dataset that consists of *spatial objects* (i.e., road objects) representing real-world roads. Several objects may represent different parts of the same real-world road, for example, each lane in a highway could be represented by a different object. Also, an object may represent more than one real-world road. An object has associated spatial and nonspatial attributes. Spatial attributes describe the location, length, shape, and topology of a road. Examples of nonspatial attributes are road number, traffic direction, number of lanes, speed limit, and so on.

3.2. Corresponding polylines

The main task in integration of spatial datasets is identifying pairs of *corresponding objects*. Corresponding objects are objects that represent the same real-world entity in distinct sources. In road maps, corresponding objects are polylines that represent the same road.

When we discuss corresponding objects in road vector data, we need to be more specific about what it means to represent a real-world road. What one person may consider as a road, a second person may consider as two roads or as part of a road. For example, if there are (1) a road from intersection point a to intersection point b and (2) a road from intersection point b to intersection point c, one person may consider this as a single road from a to c, whereas another person may consider this as two roads – one road from a to b and a second road from b to c. We solve this ambiguity using the interpretation that a road is never split by an intersection with another road. That is, we assume that roads may start or end at an intersection but should not include an intersection inside them. Thus, every polyline represents a complete road and cannot be a part of a road. In the example above, there are two roads – a road from a to b and a road from b to c. The combination of these two polylines, however, is not a road from a to c via b, because it is split by the intersection point b. This prevents the ambiguity, because it does not allow one road to be a fragment of another road, so that only undivided polylines are considered roads.

The corresponding objects should be joined in the integration. Yet, some objects may represent in one dataset a real-world entity that is not represented in the other dataset. Such objects should not be joined with any object and, thus, should not appear in any pair of corresponding objects.

We represent by *join sets* objects that should be joined. Given two datasets, a join set is one of the following two: (1) A pair of corresponding objects. (2) A single object that has

no corresponding object in the other set. We call the set of all join sets a *matching* of the spatial objects. The goal of a *matching algorithm* is to find a matching.

In many practical cases, there are no global identifiers that can tell whether two objects are corresponding objects. Hence, we must settle for an approximation when computing a matching. An approximate matching is computed according to the properties of the spatial objects. In our approach, we compute matchings based on the location of objects, because locations are always available for spatial objects, and because comparing locations can be done efficiently. Thus, using locations complies with our goal of having an efficient algorithm.

Using locations for computing a matching of polylines is not always easy. First, locations are not accurate. Thus, the same road may have different locations in different sources. Second, a polyline is represented by more than one point. Furthermore, two polylines that represent the same road may not have the same number of segments. So, there is no straightforward way of comparing all the locations of the points of two polylines for testing whether the polylines are corresponding objects. In our approach, we solve this difficulty by applying a test that for each polyline considers merely the location of the endpoints of the polyline. By considering only endpoints, each polyline is represented by two points, regardless of the number of segments it has. However, dealing with imprecise locations, different partitions to segments and incomplete information also makes endpoint-based matching an intricate task. In this article, we show how to handle this task effectively by using point-matching algorithms that were presented in the work of Beeri *et al.* (2004, 2005).

We propose a twofold integration process. Initially, a matching of the nodes is computed. Then, a matching of the polylines is generated based on the matching of the nodes. In the first phase, we say that two nodes are *corresponding* if they represent the same real-world intersection (or road end) in the two given maps. Each node has a point location. Hence, an existing algorithm (e.g., one of the algorithms in the work of Beeri *et al.* 2004, 2005) can compute an approximate matching of the nodes, based on their point locations. In principle, the join sets computed in the first phase consist of either two corresponding nodes or a single node that has no corresponding node in the other source. However, the second phase uses only the join sets that have two corresponding nodes in order to compute a matching of the polylines.

3.3. Dealing with location inaccuracy

In an integration process, the accuracy of the datasets must be taken into account, because the accuracy affects the difficulty of the problem and may affect the quality of the results. Object locations are never completely accurate. The accuracy of locations is influenced by several factors, such as the data source (land measurements, aerial photo, satellite imagery, etc.) and the techniques used to measure locations (photogrammetry, digitization, etc.), the precision of the locations in the dataset (i.e., the number of digits used for storing them), and so on. The errors in the locations of spatial objects are normally distributed, with a standard deviation σ and a mean that is equal to zero. We measure the accuracy of a dataset in terms of the *error factor* m. In this article, we assume that m is 2.5σ. When $m = 2.5\sigma$, for 98.8% of the objects in the dataset, the distance between each object and the real-world entity that it represents is less than or equal to m. Given two datasets with error factors m_1 and m_2, their *mutual error bound* is $\beta = \sqrt{m_1^2 + m_2^2}$. See the works by Mikhail and Ackermann (1982) and Lowell and Jaton (1999) for a theoretical background on error handling.

The mutual error bound is the expected maximal distance between corresponding objects. Its meaning is similar to that of the error factor. That is, in 98.8% of the cases, the distance between pairs of corresponding objects is less than or equal to β. In practice, many datasets are provided with a *metadata* describing the data source, the measurement techniques used for generating the data and the standard deviation of the error. When such metadata are not provided, we can estimate the error by using a common accuracy standard where the error has a size of 0.5 mm, or equivalently 0.02 inch, on the map (see `http://nationalmap.gov/gio/standards/`). That is, we assume an error of 0.5 mm on the scale of the map, and accordingly, we calculate the error in the real world. For instance, if the scale is 1:2000 then 0.5 mm on the map stand for 1 m in the real world.

Thus, in our algorithms, we assume that the standard deviation σ of each dataset is provided. The mutual error bound β is computed from the standard deviation values of the two input sources. Pairs of objects are candidates for being corresponding objects only if the distance between them does not exceed β. See the work by Safra *et al.* (2010) for an extended discussion about the effect of the mutual error bound on a matching of geospatial datasets.

4. Matching algorithms

We now present our algorithms for computing a matching of polylines. The algorithms receive as input two datasets, M_1 and M_2, consisting of polylines. The output is an approximate matching of the polylines. Computing the matching is a three-step process. In the first step, the algorithms find the topological nodes and generate all pairs consisting of a node and a polyline, such that the node is an endpoint of the polyline. In the second step, a matching of the nodes is computed. Finally, the matching of the polylines is generated. In this section, we discuss the details of these steps.

We propose several algorithms that are obtained from one basic algorithm by choosing (in the first step) a condition for selecting topological nodes and determining (in the second step) a semantics for computing a matching of the selected nodes. The basic algorithm AproxMatching $_{(\chi,\diamond)}$ is presented in Figure 1. Under the AND-semantics, $\diamond$ should be replaced with the logical operator `and`, that is, conjunction. Similarly, under the OR-semantics, $\diamond$ should be replaced with the logical operator `or`, that is, disjunction. The variations of the basic algorithm are also determined by considering as nodes only endpoints that satisfy a given condition χ. Possible conditions and their effects are discussed in the following section.

4.1. Finding the topological nodes

In the first step, the condition χ is applied in order to find the topological nodes that will be matched in the second phase. The condition χ selects one of the following three sets of nodes: all nodes where at least three road segments touch, all nodes where at least three road segments touch as well as all nodes where only one road object ends, and all the nodes (including nodes where only two roads meet). The following three conditions express these options.

(i) **Condition I:** The node degree is greater than 2.
(ii) **Condition II:** The node degree is different from 2, that is, is either equal to 1 or greater than 2.
(iii) **Condition III:** The node degree is any number.

AproxMatching$_{(\chi,\diamondsuit)}(M_1, M_2)$

Input: Two road maps M_1 and M_2 with a mutual error bound β
Output: A matching μ_l of the polylines in M_1 and the polylines in M_2, with respect to a condition χ on nodes and a semantics $\diamondsuit \in \{\text{AND}, \text{OR}\}$

1: **for** $i = 1, 2$ **do**
2: let S_i be all the pairs (n, l), such that n is an endpoint of polyline l in M_i
3: sort S_i according to the coordinates of the nodes
4: **for each** node n in S_i **do**
5: let *degree*(n) be the number of polylines for which n is an endpoint
6: let N_i be the set of nodes in S_i that satisfy χ
{Compute a matching μ_n of the nodes.}
7: $\mu_n \leftarrow \emptyset$
8: **for each** pair of nodes $n_1 \in N_1, n_2 \in N_2$ **do**
9: **if** $[(n_1$ is the nearest neighbor of n_2 in $N_1) \diamondsuit (n_2$ is the nearest neighbor of n_1 in $N_2)]$ and *distance*$(n_1, n_2) \leq \beta$ **then**
10: add the pair $\{n_1, n_2\}$ to μ_n
{Compute a matching μ_l of the polylines.}
11: $\mu_l \leftarrow$ **Match-Lines**(S_1, S_2, μ_n)
12: **for each** polyline $l \in M_1 \cup M_2$ **do**
13: **if** l is not in μ_l **then**
14: add the singleton $\{l\}$ to μ_l
15: **return** μ_l

Figure 1. Computing an approximate matching of polylines.

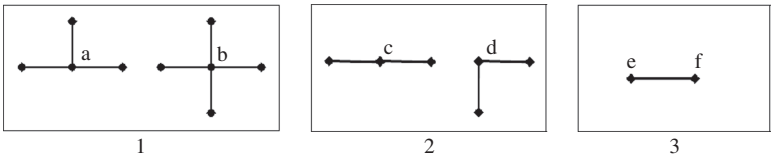

Figure 2. Endpoints and their degrees. (a) **a** and **b** are of degrees higher than 2; (b) **c** and **d** are of degree 2; (c) **e** and **f** are of degree 1.

Figure 2 shows different endpoints and their degrees. As an example, all the nodes in this figure, except **c** and **d** in Figure 2b, satisfy Condition II.

A node with degree 1 is typically a *road end*, because it does not connect different segments. A node with degree 2 is typically a *pseudo junction* and not a real intersection, because it connects two segments of the same road rather than two different roads. A node whose degree is greater than two is a real junction, because it is the intersection of at least two roads. In some maps, road ends are arbitrary points, for example, when they are caused by missing parts of roads rather than by actual road ends. Obviously, pseudo junctions can be chosen as arbitrary points on the roads, for splitting the roads into segments. Thus, matching algorithms should not always use pseudo junctions and road ends, because such nodes may not represent an entity in the real world and would not have a corresponding node in the other dataset. Therefore, under Condition I, pseudo junctions and road ends are ignored. Under Condition II, only pseudo junctions are discarded, assuming that road ends do represent real-world entities. Under Condition III, nothing is discarded, so all the nodes participate in the matching process. One of the goals of this work is to examine these conditions in order to see which one of them provides the best results in a matching algorithm.

The step of finding the relevant nodes of each dataset M_i is presented in Lines 1–6 of the algorithm of Figure 1. First, for each polyline l in M_i, where n and n' are the endpoints of l, the pairs (n, l) and (n', l) are added to the set S_i. Then, the set S_i is sorted according to the coordinates of the nodes. The sorting is according to a lexicographical order. Suppose that nodes n and n' have coordinates (x, y) and (x', y'), in correspondence. Then, a pair (n, l) will appear in the order before a pair (n', l') if and only if (1) $x < x'$ or (2) $x = x'$ and $y \leq y'$. Note that when $x = x'$ and $y = y'$, the order between the pairs is arbitrary. The sorting makes it possible to compute the degree of each node and discard nodes that do not satisfy χ, in a single pass over S_i.

Note that the step of finding the topological nodes can be done as a preprocessing in each source separately. Also, it can be computed in parallel for the two sources.

4.2. *Matching nodes*

In the second step, the algorithm of Figure 1 computes an approximate matching μ_n over the nodes of the sets N_1 and N_2 obtained in the first step. (This is true for all the variants of the algorithm.) The approximation depends on the chosen semantics. Under the AND-semantics, that is, when $\diamond$ is `and`, the algorithm finds all pairs of nodes $n_1 \in N_1$ and $n_2 \in N_2$, such that n_1 is the nearest neighbor of n_2 in N_1 and n_2 is the nearest neighbor of n_1 in N_2. The set of all such pairs is added to μ_n. This approach of matching *mutually nearest* objects was investigated in the past and is called the *mutually nearest method* by Beeri *et al.* (2004, 2005).

Note that under the AND-semantics, each node appears in exactly one pair of corresponding objects. Under the OR-semantics, that is, when $\diamond$ is `or`, the matching μ_n that the algorithm computes consists of all pairs $n_1 \in N_1$ and $n_2 \in N_2$, such that either n_1 is the nearest neighbor of n_2 (in N_1) or n_2 is the nearest neighbor of n_1 (in N_2). Note that under the OR-semantics, a node may appear in more than one pair of corresponding objects.

When matching nodes, we must take into account the error factors of the given datasets. A pair of objects that are 'too far' from each other cannot be corresponding. Hence, we compute the mutual error bound β of the sources (see Section 3.3) and under both the AND and the OR semantics, we discard from μ_n all pairs of nodes, such that the distance between them is greater than β.

One of the advantages of using the AND and the OR semantics is that they deal gracefully with datasets of very different sizes. That is, the matching is accurate even when one dataset is much larger than the other (Beeri *et al.* 2004). (Note the importance of that for typical scenarios of *ad hoc* integration, where a small dataset is matched with a larger one.) When the correspondence between objects is such that every object has at most one corresponding object in the other set, the AND semantics should be used, because each object can participate in at most one pair of corresponding objects. For example, consider two road maps M_A and M_B that correspond to the same area and such that M_B contains only the expressways in the area while M_A contains all kinds of roadways. Then, for a fragment a of an expressway in M_A, the AND semantics matches to a the corresponding fragment in M_B, and it keeps as an unmatched singleton every fragment of M_A without any corresponding fragment in the other dataset. In particular, the AND semantics suits scenarios where one dataset is a subset, or nearly a subset, of the other dataset.

When the correspondence between objects is such that an object can be matched with several objects of the other set, the OR semantics should be used. An object can be the nearest neighbor of several objects and can be matched with several objects. For example, two road maps M_A and M_B may represent the same area at different scales, such that several

lanes of an expressway are represented as several polylines in M_A and as a single polyline in M_B. In this case, the OR semantics matches several polylines of M_A that represent lane segments to a single polyline of M_B that represents the corresponding expressway segment.

Both the AND and the OR semantics compute a partial matching of the nodes, rather than a complete matching, and they only match pairs of objects when the distance between them does not exceed the mutual error bound. When deciding if two objects should be matched, the AND and the OR semantics do not consider only the distance between the objects. Instead, they check the distances of other objects to the tested pair and match only nearest neighbors. Consequently, these semantics deal gracefully with different scales and with cases where the datasets are incomplete. Furthermore, the AND and the OR semantics can be easily adjusted to cope with datasets having different levels of accuracy (Beeri *et al.* 2005) because by using an appropriate mutual error bound, the AND semantics matches to an object at most one object of the other set, and the OR semantics matches to an object o only objects that o is their nearest neighbor, regardless of the accuracy. Thus, our methods are suitable for matching datasets from maps whose scales or levels of details are far apart.

4.3. Matching polylines

In the third and final step, (each variant of) the algorithm of Figure 1 computes the matching of the polylines from the matching μ_n of the nodes. First, the method **Match-Lines**, which finds pairs of corresponding polylines, is called (Line 11 of Figure 1). Then, singletons are created from all the remaining polylines (Lines 12–14 of Figure 1).

We define four types of spatial relationships between polylines, and we consider polylines as corresponding if one of these four relationships occurs. Consider two polylines l_1 and l_2, each from a different source. Let n_1 and n_1' be the endpoints of l_1, and let n_2 and n_2' be the endpoints of l_2. The four types of relationships, which are considered as correspondence, are the following.

(i) *Complete overlap*: We say that there is a complete overlap for l_1 and l_2 if they have two pairs of corresponding endpoints, for example, both $\{n_1, n_2\}$ and $\{n_1', n_2'\}$ are corresponding nodes.
(ii) *Extension*: We say that l_1 extends l_2 when l_1 and l_2 have a pair of corresponding endpoints, and the other endpoint of l_2 is an intermediate point in l_1, for example, $\{n_1, n_2\}$ are corresponding nodes and n_2' is an intermediate point in l_1.
(iii) *Containment*: We say that l_1 contains l_2 when both endpoints of l_2 are intermediate points of l_1.
(iv) *Partial overlap*: We say that there is a partial overlap for l_1 and l_2 if each of them has an intermediate point in the other, for example, the node n_1 is an intermediate point in l_2 and the node n_2' is an intermediate point in l_1.

Figure 3 shows the four relationships between corresponding polylines.

We denote by $in(n, l)$ the predicate that is satisfied when n is an intermediate point in l and is `false` otherwise. In practice, we use an approximation when testing whether a node is an intermediate point in a polyline. Given n and l, let n' be the nearest point to n on l. Let β be the mutual error bound of the datasets, as discussed in Section 3.3. Then, $in(n, l)$ returns `true` if the distance between n and n' is not greater than β. When l contains a single segment, n' can be found by applying an orthogonal projection of n on l, and taking n' to be the nearest point to n among the calculated orthogonal projection and the two endpoints of l. When l is made of more than one segment, we first find the nearest point to

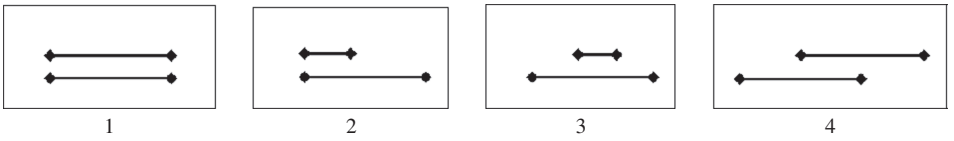

Figure 3. The four relationships between corresponding polylines. (a) Complete overlap; (b) extension; (c) containment; and (d) partial overlap.

n in each one of the segments by applying an orthogonal projection of n on each of these segments. Then, n' is the point with the shortest distance from n among the points found by the orthogonal projections and the endpoints of the segments.

The matching of the polylines according to the above four relationships is computed by the method Match-Lines presented in Figure 4. The method receives polylines and their endpoints (the sets S_1 and S_2), and a correspondence relationship μ_n for the nodes. It returns a set μ_l consisting of pairs of corresponding polylines.

The method uses two supporting data structures. A stack I is used for storing triplets consisting of a polyline, a node whose location is an intermediate point in the polyline and

Match-Lines(S_1, S_2, μ_n)

Input: Two sets S_1 and S_2 of pairs of an endpoint and a polyline, and a matching μ_n of nodes
Output: A matching μ_l of the polylines in S_1 and the polylines in S_2

1: $\mu_l \leftarrow \emptyset$, $I \leftarrow \emptyset$, $V \leftarrow \emptyset$ {μ_l is a set of pairs of matched polylines, I is a stack that keeps pairs, of a node and a line, that should be further processed, and V indicates which pairs of a node and a line were already processed.}
2: **for each** pair $\{n_1, n_2\} \in \mu_n$ **do**
3: **for each** $(n_1, l_1) \in S_1$ and $(n_2, l_2) \in S_2$ **do**
4: **if** exist $(n'_1, l_1) \in S_1$ and $(n'_2, l_2) \in S_2$ such that $n'_1 \neq n_1$ and $n'_2 \neq n_2$ and $\{n'_1, n'_2\} \in \mu_n$ **then**
5: add $\{l_1, l_2\}$ to μ_l {Complete overlap}
6: **if** exists $(n'_1, l_1) \in S_1$ such that $in(n'_1, l_2)$ **then**
7: add $\{l_1, l_2\}$ to μ_l {Extension}
8: add $(n'_1, l_2, 1)$ to I
9: **if** exists $(n'_2, l_2) \in S_2$ such that $in(n'_2, l_1)$ **then**
10: add $\{l_1, l_2\}$ to μ_l {Extension}
11: add $(n'_2, l_1, 2)$ to I
12: **while** I is not empty **do**
13: pop an element (n, l, i) from I
14: let j be the index opposite to i, *i.e.,* if i is 1 then j is 2 and if i is 2 then j is 1
15: **for each** $(n, l') \in S_i$ and $(n', l') \in S_i$ such that $in(n', l)$ **do**
16: add $\{l, l'\}$ to μ_l {Containment}
17: **if** (n', l, i) is not in V **then**
18: add (n', l, i) to I
19: **for each** $(n, l') \in S_i$ and $(n', l) \in S_j$ such that $in(n', l')$ **do**
20: add $\{l, l'\}$ to μ_l {Partial overlap}
21: **if** (n', l', j) is not in V **then**
22: add (n', l', j) to I
23: add (n, l, i) to V
24: **return** μ_l

Figure 4. Finding corresponding polylines.

the index of the source from which the node is taken. A list V is used for storing triplets as those stored in I for the purpose of recording which triplets have already been visited in the traversal of the algorithm over the nodes.

The method Match-Lines tests the existence of relationships between polylines. In Lines 2–11, it finds polylines that have a complete overlap or an extension relationship. In the case of a complete overlap, the pair of polylines is simply added to μ_l (Lines 4–5). In the case of an extension, the pair of polylines is added to μ_l and, in addition, the triplet for the polyline and the node, where the node is an intermediate point in the polyline, is added to I (Lines 6–11).

In Lines 12–22, the algorithm tries to find pairs of polylines that have a containment or a partial-overlap relationship: containment is dealt with in Lines 15–18 and partial overlap in Lines 19–22. In a run, when the algorithm reaches Line 12, stack I already contains the intermediate points that were discovered during the search for polylines having the extension relationship. When new intermediate points are discovered, they are added to I. The list V is used for recording which intermediate nodes have already been visited, as part of a bookkeeping intended to make sure that we do not process the same intermediate node more than once. Note that Match-Lines may not discover all the polylines that have a containment or a partial-overlap relationship, because in the traversal over the nodes, the algorithm does not visit intermediate nodes that are isolated, that is, nodes that are not connected by an edge to a visited node. This is done on purpose, because our goal is to provide an approximate matching while keeping the algorithm efficient.

When one of the matched datasets is incomplete or when the datasets have different scales, a road can be represented in one dataset, by some polyline l, without being represented in the other dataset. In such a case, the polyline l does not have a corresponding polyline in the other dataset, and indeed, Match-Lines is not likely to match l to any polyline. To see why, recall that under the AND semantics, in order to match l to some polyline l' with complete overlap, there should be an endpoint of l and an endpoint of l' that are the nearest neighbors of each other; moreover, the other two endpoints of l and l' should also be the nearest neighbors of each other. In most cases, the corresponding polyline of l' will be matched to l', preventing l from being matched to l' (because each endpoint of l' has only one nearest neighbor in the other dataset). However, even if l and l' are matched (and our experiments show that this happens in a relatively few cases), the distance between the corresponding endpoints will not be greater than the mutual error bound. This requires l and l' to be approximately parallel lines that are very close to each other, unless they are shorter than the error bound. So, even erroneous matchings are reasonable. Similar arguments can be applied to the other spatial relationships (extension, containment, and partial overlap), and in the case where the matching is constructed under the OR semantics. Hence, Match-Lines handles well incomplete datasets and scale differences.

4.4. Data structures

In order to improve the efficiency of the computation, two auxiliary data structures, in addition to I and V, are used in the implementation of Match-Lines. One data structure, denoted by L_{point}, provides for each point p a list of all the polylines having p as an endpoint. The second data structure, denoted by L_{polyline}, stores for each polyline its two endpoints.

The two data structures are implemented as a Vector. (A Vector is similar to an array except that its size can grow.) Each node of the input datasets is mapped to a unique number (thus, the nodes are identified as $1, 2, 3, \ldots$). Hence, given a node i, we can directly access

the ith entry of L_{point} in $O(1)$ time in order to get a pointer to the list of polylines that have i as an endpoint. Polylines are mapped to numbers in a similar way, so retrieving the endpoints of a polyline from L_{polyline} can also be done in $O(1)$ time.

4.5. *Dealing with length anomalies*

There are two anomalous cases in which matching polylines based on merely the matching of the endpoints is problematic. In this section, we present these two cases, which we have encountered during our experiments. Also, we explain how our algorithms can be easily modified to deal with these cases. In fact, the anomalies we consider in this section are quite rare in all the datasets we tested. Thus, our algorithms are accurate even without the modifications hereafter. We will discuss the effect of these modifications on the accuracy of the algorithm in Section 7.

4.5.1. *Short polylines*

The first anomaly we encountered is due to short polylines. We say that a polyline is *short* if its length is smaller than the error bound of its source. When the two endpoints of a short polyline l are near some intersection X of polylines, it may happen that the two endpoints of l will be considered as intermediate points of several polylines that intersect in X. If this happens, l may be matched to polylines that have different orientations. Users often consider such a matching as an error. For example, consider the roads in Figure 5. Suppose that Polyline 2–4 (i.e., the polyline between Points 2 and 4) is short. In this case, Point 2 and Point b will be deemed corresponding. Point 4 will be considered an intermediate point on each of the three polylines: Polyline `a-b`, Polyline `b-c`, and Polyline `b-d`. Thus, Polyline 2–4 will be deemed contained in all the three polylines – Polyline `a-b`, Polyline `b-c`, and Polyline `b-d`. Yet, we would like Polyline 2–4 to be considered as contained only in Polyline `b-d`.

For correctly handling short polylines, we define for each one of them a new error bound that is equal to a half of its length. Then, we compute for each pair of a polyline and a short polyline, a new mutual error bound. For instance, if we try to match a polyline l_1 from a source having an error factor m_1 with a short polyline l_2, the mutual error bound of the two polylines is the following.

$$\beta = \sqrt{m_1^2 + \left(\frac{\text{length}(l_2)}{2}\right)^2}$$

As we discussed earlier, if an endpoint n of l_2 has a distance from l_1 that is greater than β, then n cannot be an intermediate point of l_1. This helps to ensure that matched polylines will have the same orientation, while avoiding complex geometric computations. If, for

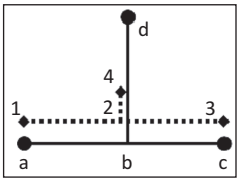

Figure 5. Polyline 2–4 will be matched to Polyline `a-b`, Polyline `b-c`, and Polyline `b-d`.

example, we use this approach in the case presented in Figure 5, then Point 4 will only be considered an intermediate point of Polyline b-d.

4.5.2. Length differences

The second special case we encountered is when two polylines have the same endpoints, yet, the curvature of the lines is different. This happens, for instance, when one polyline is straight while the other is curved. For example, in Figure 6, Polyline a-b goes straight from Point a to Point b while Polyline 1–2 connects the points in a curved route and, thus, is much longer than Polyline a-b. When considering only the endpoints, our algorithm will assert that the two polylines are corresponding. However, many people will consider this assertion as an error. A similar error may occur when trying to match a straight polyline to a curved line where one or both of the endpoints, of the curved polyline, are intermediate points on the straight polyline (see an example in Figure 7).

In order to discover anomalies of this type, we compare the lengths of matched polylines. If the ratio of the lengths (i.e., the length of the shorter polyline divided by the length of the longer polyline) is below some threshold t, then the two polylines are not considered corresponding. We tested several threshold values and found that $t = 0.5$ effectively discarded incorrect matches without discarding correct matches. Note that when the endpoints of one polyline are intermediate points of the other polyline, we use the length of projecting the first polyline onto the second to avoid the erroneous assertion of a containment relationship. An example of such a projection is depicted in Figure 8, where Points 2 and 3 are intermediate points of Polyline a-d, and hence, we divide the length of the projection of Polyline 2–3 onto Polyline a-d by the length of Polyline a-d.

Discovering short polylines or length differences requires knowing the lengths of polylines. This is not a problem, because in many geospatial information systems, the length of each polyline is computed once and then stored in the system. Thus, in order to apply our techniques to deal with length anomalies, there is no need to do complex geometric computations, and there is no significant influence on the efficiency of the algorithms.

4.6. *Using node degrees for improving the algorithms*

In Section 4.2, we discussed the discovery of corresponding nodes, and we considered methods that are based merely on the locations of the nodes. However, Safra *et al.* (2006)

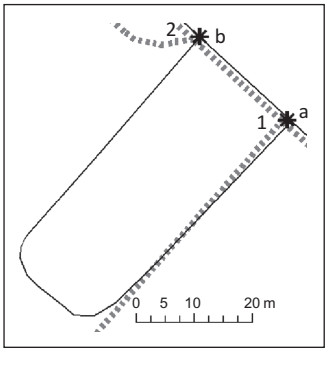

Figure 6. Polyline 1–2, which has the shape of a cup (⊔), should not be matched to Polyline a-b due to length differences.

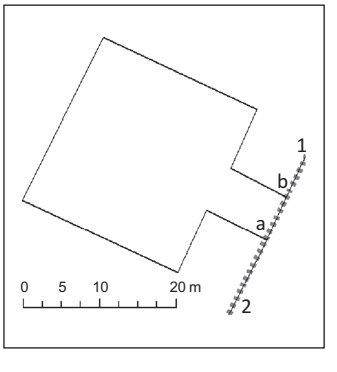

Figure 7. Polyline a-b, which has the shape of a tennis racket, should not be considered as contained in Polyline 1–2 due to length differences.

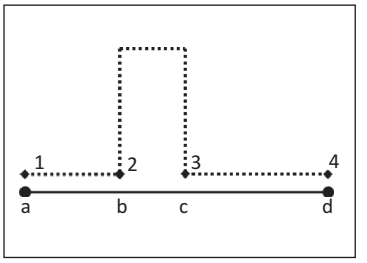

Figure 8. The interval b-c on Polyline a-d is the projected part of Polyline 2–3 on Polyline a-d.

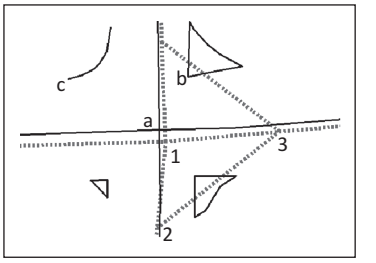

Figure 9. Maps in which node degrees can improve the matching.

showed that combining locations with additional information, when computing a matching, may improve the quality of the result. Particularly, we can use the degree of nodes in the road network. It is local information that can be computed for each node by merely counting the neighbors of the node, while ignoring complex topological properties of the entire network, such as the number of connected components and articulation points.

When using node degrees, we can simply filter out, from the matching of the nodes, pairs of nodes that have different degrees. Note that such filtering does not affect the generality of our approach. We will later show that the filtering may improve the quality of the result and increase the efficiency of the computation. However, our methods are efficient and effective even without this filtering.

The filtering step is illustrated in the following example. Figure 9 shows two maps that contain several nodes having different degrees. (The roads in Figure 9 are fragments of two real maps that we have used in our experiments.) In these maps, only the two nodes whose degree is four (Node a and Node 1) should be matched. Thus, we would like the algorithm to discard matches such as that of Node b and Node 1. Note that such a matching may occur when using the OR-semantics.

As data sources are sometimes heterogeneous and may represent incomplete information, we allow some level of flexibility in the removal of matched pairs. To do so, we let users provide a threshold, and we discard from the matching only pairs of nodes, such that the difference between their degrees exceeds the threshold. The threshold should be chosen according to the heterogeneity of the sources.

Removing pairs of nodes during the first step of the algorithm affects both the efficiency of the algorithm and the accuracy of the result. The removal decreases the number of matchings that the algorithm needs to examine and thus reduces the running time of the algorithm. Therefore, efficiency is improved. If most of the discarded pairs are erroneous matches, the removal increases the accuracy of the algorithm, because it improves the accuracy of the first step. Obviously, a removal of correct pairs may reduce the accuracy.

4.7. Time and space complexity

In this section, we analyze the time and space complexities of the method AproxMatching $_{(\chi,\diamond)}$. (In the analysis, we use common terms and notations, see Cormen *et al.* (2009).) Suppose that the input consists of two road maps M_1 and M_2, and let k_1 and k_2 be the number of polylines in M_1 and M_2, respectively. Note that in this case, the number of nodes in M_1 is at most $2k_1$, and in M_2 it is at most $2k_2$.

In the first step of the algorithm (i.e., finding the topological nodes), all the operations, except for the sort, have a linear time complexity in the size of the input. Thus, the time complexity of the first step is $O(k_1 \log k_1 + k_2 \log k_2)$, which is the complexity of the sort. The space complexity is $O(k_1 + k_2)$.

When computing the matching of the nodes, the nearest-neighbor function is used. Suppose that we use an implementation of the nearest-neighbor function that has the following time and space complexities. For a given point and a set of k points, the function finds the nearest neighbor of the point in time complexity $T_{nn}(k)$ and in space complexity $S_{nn}(k)$. Then, under the AND-semantics, the time complexity of matching the nodes is either $O(k_1 T_{nn}(k_2))$ or $O(k_2 T_{nn}(k_1))$, depending on the dataset that we iterate on. The space complexity is $O(\min\{k_1, k_2\} + S_{nn}(k_1) + S_{nn}(k_2))$, because the number of mutually nearest neighbors is at most $\min\{k_1, k_2\}$. Under the OR-semantics, the time complexity is $O(k_1 T_{nn}(k_2) + k_2 T_{nn}(k_1))$. The space complexity is $O(k_1 + k_2 + S_{nn}(k_1) + S_{nn}(k_2))$, because the number of pairs in which one node is the nearest neighbor of the other is at most $k_1 + k_2 - 1$.

For the part of computing the matching of the polylines by the method Match-Lines, a rough estimation of the time complexity is $O((k_1 k_2)^2 \log(|\mu_n|))$, where $|\mu_n|$ is the number of corresponding nodes, which is at most $\min\{k_1, k_2\}$ under the AND-semantics and $k_1 + k_2 - 1$ under the OR-semantics. The space complexity is $O(k_1 + k_2)$ under both semantics, because of the data structures that the algorithm maintains.

For a finer estimation of the time complexity of Match-Lines, we assume that d is the maximal degree of nodes in M_1 and M_2. First, we analyze the complexity of testing overlap and extension. In the test, sets of polylines with a shared node are matched against sets of polylines that also have a shared node, where the sets are from different sources and the shared nodes are corresponding nodes. Each such matching attempt is over two sets whose size is not greater than d. These matching attempts are done for each pair of corresponding nodes. The number of corresponding nodes is the size of the set μ_n. Hence, the time complexity of testing overlap and extension is $O(d^2 |\mu_n|)$.

When containment and partial overlap are tested, nodes are popped out of I iteratively – each node at most once. Recall that there are at most $2(k_1 + k_2)$ nodes in M_1 and M_2. Then, at most d pairs of a node n' and a polyline l' are considered for each popped node, in the iterations of Line 15 and of Line 19 of Match-Lines. Retrieving the polylines that the popped node is their endpoint can be done in time logarithmic in the sizes of the sets S_1 and S_2. Thus, the time complexity for these tests is $O(k_1(\log k_2 + d) + k_2(\log k_1 + d))$. Preventing the algorithm from processing the same triplet twice is by checking whether the triplet is in V before inserting it into I. There are at most $2(k_1 + k_2)$ elements in V, so this test has $O(\log(k_1 + k_2))$ time complexity.

The following proposition summarizes the analysis of the time and space complexities.

Proposition 4.1 *Let M_1 and M_2 be road maps containing k_1 and k_2 polylines, respectively, and suppose that $k_1 \geq k_2$.*

(i) *When called with M_1 and M_2, the time complexity of the method AproxMatching$_{(\chi,\text{AND})}$ is $O(k_1(\log k_1 + d) + k_2 d^2 + \min\{k_1 T_{nn}(k_2), k_2 T_{nn}(k_1)\})$.*
(ii) *The time complexity of AproxMatching$_{(\chi,\text{OR})}$ on M_1 and M_2 is $O(k_1(\log k_1 + d^2) + k_1 T_{nn}(k_2) + k_2 T_{nn}(k_1))$.*
(iii) *The space complexity under both the* AND *and the* OR *semantics is $O(k_1 + S_{nn}(k_1))$.*

Note that Walter and Fritsch (1999) have proposed a matching algorithm that has exponential time complexity while the time complexity of our algorithms is almost linear.

5. An alternative approach of using a spatial index for matching road networks

The algorithm AproxMatching$_{(\chi,\diamond)}$ of Figure 1 finds the matching μ_l of the polylines by first matching every two polylines that have (at least) one pair of matching endpoints. Then, it continues to match polylines by means of the triplets that are popped from I. To work efficiently, AproxMatching$_{(\chi,\diamond)}$ needs only simple (nonspatial) data structures (namely, L_{point} and L_{polyline}).

An alternative to AproxMatching$_{(\chi,\diamond)}$ is a conceptually simple algorithm that uses a spatial index. The basic idea is to consider all pairs (l_1, l_2) of polylines, where l_1 is from one dataset and l_2 is from the other. For each pair (l_1, l_2), we should test whether it satisfies one of the four relationships illustrated in Figure 4. In order to reduce the number of pairs that have to be considered, we can use a spatial index on one of the two datasets. In a typical scenario of *ad hoc* retrieval, a user provides a small fragment of a road network to be matched to a large road network. In this case, a spatial index on the larger source will substantially reduce the running time. In this section, we present two methods of using a spatial index on one of the datasets, thereby reducing the numbers of pairs that have to be considered.

Both the methods use a grid index (mesh) as follows: Consider two road maps M_s and M_l. We assume that M_s is the road network that a user provides for matching against a road map M_l. We also assume that a grid index exists for M_l. (Typically, M_l is the larger map among M_s and M_l; however, our approach works also for the case where M_l is smaller than M_s.) The grid index is constructed by dividing the area of M_l into square cells. For each cell, the index holds all the polylines that intersect the area of the cell. Now, for some given area A, the polylines of M_l that intersect A are retrieved in a twofold process. First, the cells that intersect the bounding box of A are computed, and the polylines in these cells are retrieved. Then, for each retrieved polyline, we check whether it indeed intersects A.

In the following sections, we present two ways of using the grid index for retrieving the relevant polylines. In the two methods, a buffer is constructed around each element (endpoint or polyline) of M_s and the index is used for retrieving the polylines of M_l that intersect the buffer. The difference between the two methods is in whether buffers are constructed with respect to the endpoints of polylines or with respect to entire polylines. In Section 7.5, we show that the algorithm AproxMatching$_{(\chi,\diamond)}$ is much more efficient than the two methods that use a grid index.

5.1. *Endpoint buffers*

In the *endpoint-buffer method*, a buffer is generated around each endpoint of every polyline of M_s, and the grid index is used for retrieving the polylines of M_l that intersect the buffer. The buffer has a circular shape with a radius that is equal to the mutual error bound β. For each polyline p_s of M_s, there are two relevant buffers – a buffer for each endpoint. Note that a polyline p_l of M_l that does not intersect the buffers of a polyline p_s cannot have a complete overlap, partial overlap, or extension correspondence with p_s. Furthermore, p_l also cannot contain p_s, but it can be contained in p_s. Thus, after the step of retrieving polylines that intersect the buffers of p_s, there is a need to retrieve the polylines that are candidates for being contained in p_s. This is done by generating a minimum bounding rectangle (MBR) around p_s, and enlarging its width and length by 2β, symmetrically (i.e., let

(x_1, y_1) and (x_2, y_2) be the points that define the bounding box, then the enlarged bounding box is defined by the points $(x_1 - \beta, y_1 - \beta)$ and $(x_2 + \beta, y_2 + \beta)$). All the polylines of M_l whose two endpoints are contained in the enlarged MBR are retrieved and tested for containment in p_s.

5.2. *Polyline buffers*

In the *polyline-buffer method*, a buffer is generated around each polyline p_s of M_s, and polylines of M_l that intersect the buffer are retrieved. The buffer of p_s is the area of all the points whose distance from p_s is less than or equal to β. The polylines of M_l that intersect the buffer are retrieved using the grid index and then they are tested to check whether indeed they correspond to p_s.

6. Measuring the quality of the results

As in information retrieval, we measure the quality of a matching algorithm in terms of *recall* and *precision*. In this section, we discuss four measures of recall and precision that we used in our experiments.

The *basic* definition of recall and precision measures the rate of correct join sets. Recall is the percentage of correct sets that actually appear in the result (e.g., 87% of all the correct sets appear in the result). Precision is the percentage of correct sets out of all the sets in the result (e.g., 92% of the sets in the result are correct).

Consider an integration of two maps, where C denotes the set of correct join sets appearing in the result, A is the set of all the correct join sets, and R is the set comprising all the sets in the result. Then, in the basic definition, the recall is $\frac{|C|}{|A|}$ and the precision is $\frac{|C|}{|R|}$.

An alternative measure, called *pair count,* is that of counting only pairs and ignoring singletons. Suppose that C_p, A_p, and R_p are obtained by discarding the singletons from C, A, and R, respectively. Then, the recall is $\frac{|C_p|}{|A_p|}$ and the precision is $\frac{|C_p|}{|R_p|}$.

The following example illustrates the differences between the two measures. Consider two datasets: M_A with polylines a_1, a_2, and a_3, and M_B with polylines b_1, b_2, b_3, and b_4. Suppose that the correct matching comprises the three pairs (a_1, b_1), (a_2, b_2), (a_3, b_3), and the singleton (b_4). Then, a result $R_1 = \{(a_1, b_1), (a_2, b_2), (a_2, b_3), (a_3), (b_4)\}$ has a recall of $\frac{3}{4}$ and a precision of $\frac{3}{5}$, when singletons are considered. The recall is $\frac{3}{4}$ because R_1 contains two correct pairs and one correct singleton, of the four items in the correct matching. The precision is $\frac{3}{5}$ because among the five items of R_1, only the three items $(a_1, b_1), (a_2, b_2)$, and (b_4) are correct. When using the pair-count measure, we ignore the singletons. Thus, the recall is $\frac{2}{3}$, because there are three pairs in the correct matching and two of them appear in R_1. The precision is $\frac{2}{3}$, because two pairs, among the three pairs of R_1, are correct.

The pair-count measure should be used when the quality of the result depends only on the number of pairs that were matched. The basic definition should be used when correctly identifying the singletons is significant. For instance, consider an integration of an old map and a new map. It is likely that a road, in the new map, that does not have a corresponding road in the old map is a new road. If it is important to know which roads are new, one should use a method that is accurate according to the basic measure.

There are cases where we want methods to be influenced by the lengths of the polylines that are matched: a pair of long roads should have a greater influence on the recall and precision than a pair of short roads. In such cases, we use *length-based* recall and precision. The length of a pair of polylines is defined as the length of the overlapping part of the

polylines. For a singleton, the length of the set is the length of the single object. The length-based measure may be viewed as a *weighted* measure (similarly to weighted average), where the weight is according to the length. Then, in the basic length-based measure, the recall r and the precision p are defined as follows:

$$r = \frac{\sum_{s \in C} \text{length}(s)}{\sum_{s \in A} \text{length}(s)} \tag{1}$$

$$p = \frac{\sum_{s \in C} \text{length}(s)}{\sum_{s \in R} \text{length}(s)} \tag{2}$$

A fourth measure is obtained by using C_p, A_p, and R_p instead of C, A, and R, respectively, in Equations (1) and (2).

By and large, methods that provide high recall and precision according to length-based measures are good for integrating road maps of rural areas, that is, maps where long roads are more important than short roads. Methods that provide high recall and precision in measures based on counting join sets are suitable for integrating road maps of urban areas, that is, maps that contain many short roads and the importance of a road does not depend on its length.

In our tests, we used all the four measures for comparing our methods with the result of an integration performed by a human expert. That is, the join sets found by the expert form the set C of all the correct join sets, and we computed our measures with respect to C.

7. Experiments

In this section, we describe our experiments for determining the efficiency and accuracy of the six variants of the basic algorithm AproxMatching$_{(\chi,\diamond)}$ of Figure 1. We use and to denote the two semantics of Section 4.2 for matching nodes. The numerals 1, 2, and 3 indicate the three conditions of Section 4.1 that determine the nodes participating in the matching process. Altogether, we tested six algorithms; for example, 'AND 1' denotes the algorithm that selects the nodes according to the first condition (i.e., the degree is greater than 2) and uses the AND-semantics for matching them.

The experiments were aimed at showing the efficiency and the accuracy of our algorithms, and in particular answering the following three questions.

(i) Which of the three conditions of Section 4.1 is best for selecting nodes?
(ii) Which of the two semantics of Section 4.2 gives better results?
(iii) What is the effect of the improvements that were discussed in Sections 4.5 and 4.6.

We considered these questions with respect to the different ways of measuring the quality of the result (i.e., length vs. sets).

7.1. The datasets we used

In our experiments, we used real-world datasets from maps of three different cities: New York City (NY, USA), Tel Aviv (Israel), and Haifa (Israel). Tel Aviv and New York are located in relatively flat areas while Haifa is built on a hilly area. The different datasets were collected by different organizations, at different times, and using different collection methods.

7.1.1. New York datasets

We used two maps of New York in our tests. The first map is published on the World Wide Web by the Department of City Planning of New York[1] and we refer to this dataset as LION. The Web site does not specify the accuracy of this map. Yet, it does say that the accuracy of the data in LION has been gained by spatially aligning the features of the map with an aerial photograph.

The second map we used was taken from Cornell University Geospatial Information Repository[2] (the Census 2000 version). We refer to this dataset as CUGIR. The source of the data in CUGIR is the Census TIGER (Topologically Integrated Geographic Encoding and Referencing) database. The accuracy level of CUGIR complies with the standard of the US Geological Survey (USGS) for 1:100,000-scale maps.

We extracted from LION and CUGIR the part that represents Manhattan and used it for the tests. The dataset we extracted from LION contains 1141 polylines. The dataset we extracted from CUGIR contains 555 polylines. Fragments of the maps are presented in Figure 10.

In Table 1, we show for each of the six variants of the algorithm, the number of nodes that were created from each dataset. Also, we present the number of pairs of corresponding nodes that were found in the second step of the algorithm. The algorithm results were compared with a *correct matching*, that is, a matching that was determined manually by a human expert. The correct matching consists of 865 join sets: 752 pairs and 113 singletons having total lengths of 87,651 and 5933 m, respectively.

7.1.2. Tel Aviv datasets

A second pair of datasets that we used in the tests was extracted from road maps of the city of Tel Aviv. One dataset was collected by the Survey of Israel,[3] and we refer to it as SOI. This dataset was extracted from aerial photographs at the scale of 1:40,000 (equivalent to digital maps at the scale of 1:5000–1:10,000). The other dataset we used was collected by a

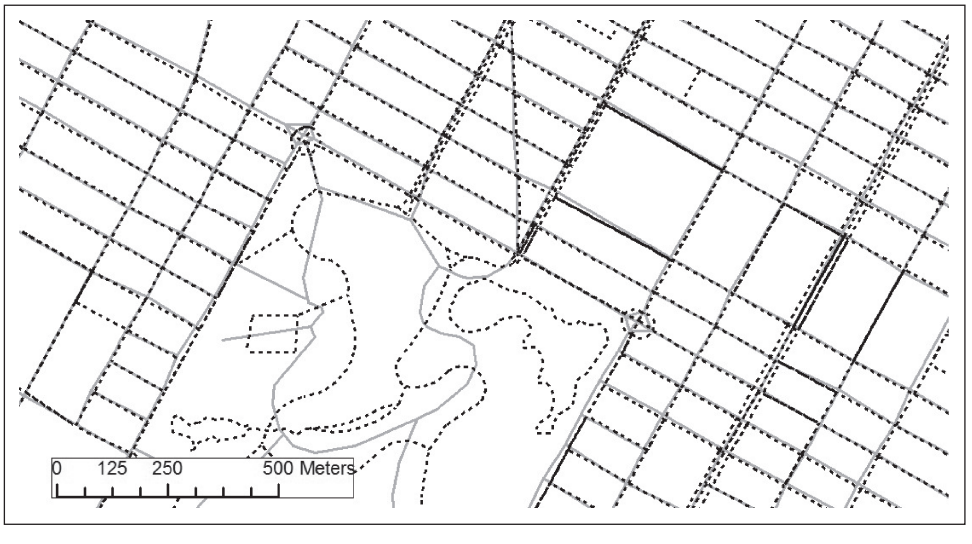

Figure 10. Fragments of the CUGIR (dashed lines) and LION (solid lines) road maps (Manhattan).

Table 1. The number of nodes and the number of pairs of nodes in the New York datasets.

	Number of nodes in CUGIR	Number of nodes in LION	Number of pairs
AND 1	353	255	251
OR 1			305
AND 2	356	261	254
OR 2			319
AND 3	386	291	279
OR 3			359

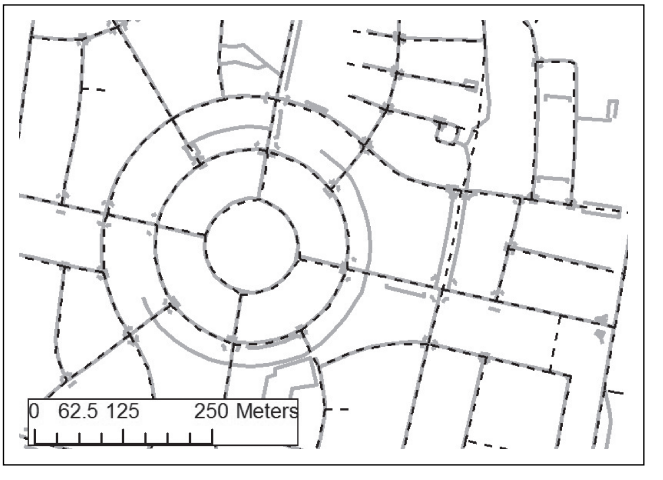

Figure 11. Fragments of the SOI (solid lines) and MAPA (dashed lines) road maps (Tel Aviv).

commercial corporation,[4] which we refer to as MAPA. This dataset was extracted directly from a digital map, at the scale of 1:25,000, by the Mapa Corp. The dataset of SOI contains 404 polylines. The dataset of MAPA contains 165 polylines.

Part of the test area can be seen in Figure 11. Although the two maps have a similar scale, there is a large difference in how they describe the same road network. The reason for the large difference is that while SOI is oriented to the generation of maps and other geometric measurements, MAPA is used mainly for road navigation. To illustrate the difficulties that this difference may cause during an integration, Figure 12 shows a small vicinity of two junctions. It can be seen that SOI has some isolated polylines, whereas in MAPA all the polylines form a connected network.

In Table 2, we show for each of the six variants of the algorithm, the number of nodes that were created from each dataset (in the first step of the algorithm) and the number of corresponding pairs of nodes that were found in the second step. The correct matching that we determined manually consists of 274 join sets: 204 pairs and 70 singletons, having total lengths of 14,483 and 4934 m, respectively.

Determining the correct join sets was not always straightforward. For example, in Figure 12, it is not clear if the short roads from SOI should be matched to zero, one or two objects of MAPA. In such cases, those roads were not included in any correct join set, and in the result, join sets containing those roads were removed.

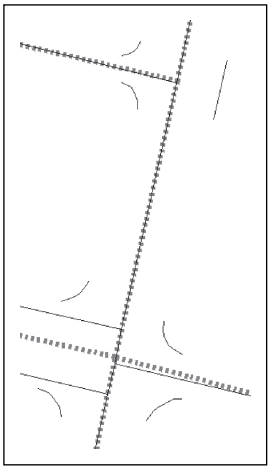

Figure 12. An enlarged view of the vicinity of two junctions.

Table 2. The number of nodes and the number of pairs of nodes in the Tel Aviv datasets.

	Number of nodes in SOI	Number of nodes in MAPA	Number of pairs
AND 1	100	83	67
OR 1			84
AND 2	513	111	93
OR 2			451
AND 3	545	124	101
OR 3			486

7.1.3. Haifa datasets

We used the SOI and MAPA data sources for tests on maps of the city of Haifa. For Haifa, the dataset of SOI contains 724 polylines, and the dataset of MAPA contains 640 polylines. Differently from New York and Tel Aviv, Haifa is located in a hilly area. Thus, the roads in Haifa are more curved than in the other two cities, and the maps of Haifa are somewhat more chaotic than the maps of New York or Tel Aviv. (See Figure 13 for a visual view of the maps.) This increases the intricacy of the integration.

Table 3 shows, for each of the six variants of the algorithms, the number of nodes that were created from each dataset (in the first step of the algorithm) and the number of corresponding pairs of nodes that were found in the second step. The manually determined correct matching consists of 720 join sets: 484 pairs and 236 singletons having total lengths of 32,766 and 19,042 m, respectively.

7.1.4. The error factors of the sources

First, we consider the error factors of the New York datasets. The map of TIGER is of scale 1/24,000. We assume a standard error of 0.25 mm in the map. According to the

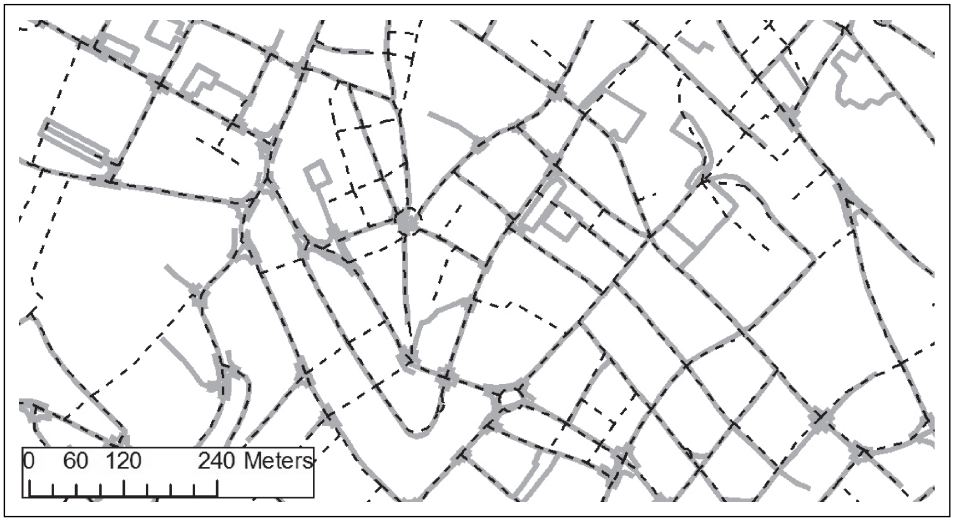

Figure 13. Fragments of the SOI (solid lines) and MAPA (dashed lines) road maps (Haifa).

Table 3. The number of nodes and the number of pairs of nodes in the Haifa datasets.

	Number of nodes in SOI	Number of nodes in MAPA	Number of pairs
AND 1	343	236	166
OR 1			241
AND 2	888	423	233
OR 2			779
AND 3	905	458	248
OR 3			821

scale, we get a standard deviation σ of 6 m in the real world. Thus, the error factor is $m = 2.5\sigma = 15$ m. For LION, the only information we have is that the accuracy of the data has been gained by spatially aligning the features of the map with an aerial photograph. We assumed from the detail level of the map that the aerial photograph is of the same scale as in TIGER. Using such a photograph, a map of scale 1/5000–1/10,000 is produced. So, we estimated a scale of 1/7500. With that scale, an error of 0.25 mm in the map yields an error of 1.88 m in the real world. As the map was aligned with the photograph and was not produced by a direct mapping of features to the photograph, we increased the error by a factor of 2. So, we used a σ of 3.76 m, and an error factor $m = 2.5\sigma$ that is equal to 9.4 m. Therefore, the mutual error bound for TIGER and LION is $\beta = \sqrt{15^2 + 9.4^2} = 17.7$ m.

For the datasets of Tel Aviv and Haifa, we used the following error factors. SOI is provided with a specified σ of 2 m; hence, the error factor m is 5 m. MAPA was digitized from a map of scale 1/25,000. We assume a standard error of 0.5 mm in the map and the digitization process adds to the error another 0.2 mm. So, the error in the map is $\sqrt{0.5^2 + 0.2^2} = 0.29$ mm. Having the scale of 1/25,000, the standard deviation σ of the error in the real world is 7.25 m and we rounded it to 8 m. The error factor m is 2.5σ, that is, 20 m. Now, with an error factor of 5 m in one source and an error factor of 20 m in the other source, $\beta = \sqrt{5^2 + 20^2} = 20.6$ m.

In order to verify our calculations, we conducted our tests using several different error factors, in addition to the error factors that are presented in this section. Inall our tests, using the different error factors never provided better results than when using the error factors we presented here.

7.2. Test results

We now present the results of our experiments over the road maps of New York, Tel Aviv, and Haifa. In Section 7.3, we will explain and analyze these results.

Tables 4–6 show, for each city, the numbers of pairs and singletons that were produced in the integration, and the numbers of the correct join sets in the result. Tables 7–9 and Figures 14–16 present the recall and precision of each algorithm, using the two methods for measuring the quality of the result. Tables 10–12 and Figures 17–19 also show the recall and precision of each algorithm, however, only for pairs; that is, each matching consists of merely pairs and the singletons are ignored.

Table 4. The total number of join sets and the number of correct ones in the result (New York).

	Pairs in the result	Singletons in the result	Correct pairs	Correct singletons
AND 1	651	161	647	101
OR 1	721	115	684	92
AND 2	651	161	647	101
OR 2	724	111	687	92
AND 3	650	156	646	103
OR 3	734	90	693	84

Table 5. The total number of join sets and the number of correct ones in the result (Tel Aviv).

	Pairs in the result	Singletons in the result	Correct pairs	Correct singletons
AND 1	187	79	186	69
OR 1	225	72	190	67
AND 2	189	73	186	65
OR 2	235	55	193	54
AND 3	191	76	186	67
OR 3	264	50	193	49

Table 6. The total number of join sets and the number of correct ones in the result (Haifa).

	Pairs in the result	Singletons in the result	Correct pairs	Correct singletons
AND 1	479	237	452	217
OR 1	575	207	460	201
AND 2	474	234	448	211
OR 2	543	199	457	193
AND 3	468	220	458	211
OR 3	558	195	460	191

Table 7. Recall and precision (New York).

	Recall length	Precision length	Recall sets	Precision sets
AND 1	0.92	0.95	0.86	0.92
OR 1	0.95	0.97	0.90	0.93
AND 2	0.92	0.95	0.86	0.92
OR 2	0.95	0.98	0.90	0.93
AND 3	0.92	0.96	0.87	0.93
OR 3	0.95	0.99	0.90	0.94

Table 8. Recall and precision (Tel Aviv).

	Recall length	Precision length	Recall sets	Precision sets
AND 1	0.98	0.96	0.99	0.96
OR 1	0.99	0.95	0.99	0.87
AND 2	0.95	0.95	0.97	0.96
OR 2	0.94	0.94	0.96	0.85
AND 3	0.96	0.94	0.98	0.95
OR 3	0.93	0.91	0.94	0.77

Table 9. Recall and precision (Haifa).

	Recall length	Precision length	Recall sets	Precision sets
AND 1	0.92	0.89	0.96	0.93
OR 1	0.91	0.89	0.95	0.93
AND 2	0.91	0.89	0.95	0.93
OR 2	0.90	0.90	0.93	0.88
AND 3	0.93	0.92	0.96	0.95
OR 3	0.90	0.90	0.93	0.86

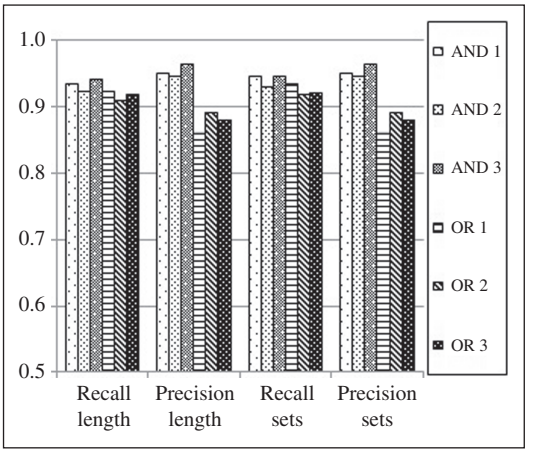

Figure 14. Recall and precision, considering pairs and singletons (New York).

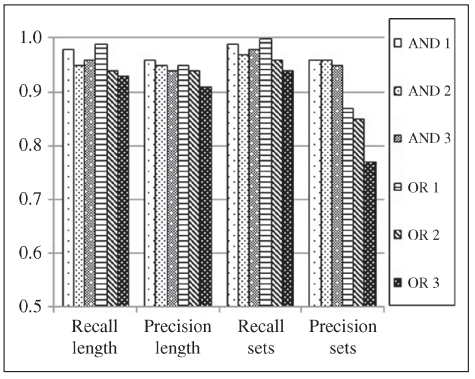

Figure 15. Recall and precision, considering pairs and singletons (Tel Aviv).

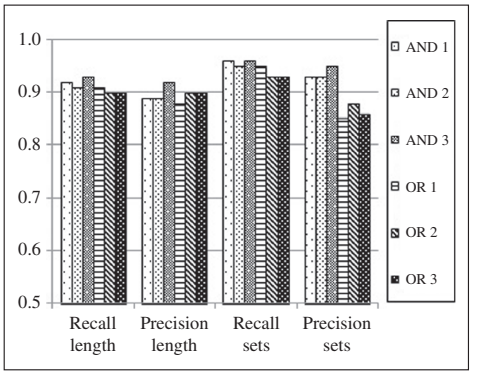

Figure 16. Recall and precision, considering pairs and singletons (Haifa).

Table 10. Recall and precision, pairs (New York).

	Recall length	Precision length	Recall sets	Precision sets
AND 1	0.92	1.00	0.86	1.00
OR 1	0.95	0.99	0.91	0.95
AND 2	0.92	1.00	0.86	1.00
OR 2	0.96	0.99	0.91	0.95
AND 3	0.92	1.00	0.86	1.00
OR 3	0.96	0.99	0.92	0.94

Table 11. Recall and precision, pairs (Tel Aviv).

	Recall length	Precision length	Recall sets	Precision sets
AND 1	0.98	0.99	0.99	0.99
OR 1	0.99	0.95	0.99	0.84
AND 2	0.98	0.99	0.99	0.98
OR 2	0.99	0.92	0.99	0.82
AND 3	0.98	0.98	0.99	0.97
OR 3	0.99	0.90	0.99	0.73

Next, we will show how the results are influenced by the modifications for dealing with length anomalies, which are presented in Section 4.5. There were no length anomalies during the integration of either the New York or Tel Aviv datasets. However, over the Haifa datasets length anomalies caused 10 incorrect matches having a total length of 1345 m. When we applied our techniques for dealing with the anomalies, these erroneous matches were removed from the result. Accordingly, 10 singletons were added to the result with a total length of 1694 m. The results of the algorithms, when dealing with length anomalies, are shown in Tables 13 and 14 and Figures 20 and 21. Note that when considering pairs and singletons, both recall and precision are improved by the modifications. Yet, when considering only pairs, just the precision is improved because merely removing pairs cannot increase the recall.

Table 12. Recall and precision, pairs (Haifa).

	Recall length	Precision length	Recall sets	Precision sets
AND 1	0.95	0.92	0.98	0.94
OR 1	0.98	0.87	0.99	0.89
AND 2	0.95	0.92	0.97	0.95
OR 2	0.98	0.89	0.99	0.84
AND 3	0.98	0.92	0.99	0.94
OR 3	0.99	0.88	0.99	0.82

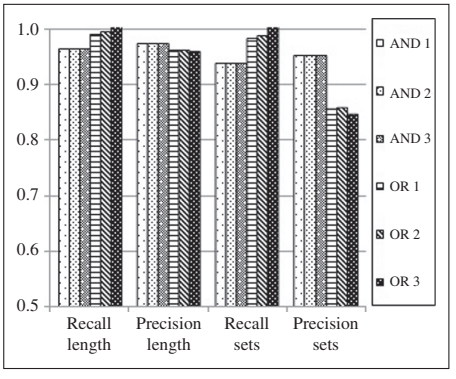

Figure 17. Recall and precision, considering just pairs (New York).

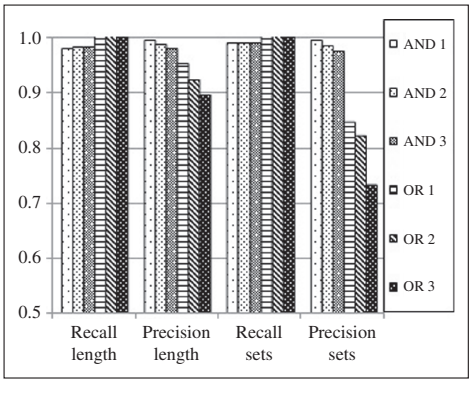

Figure 18. Recall and precision, considering just pairs (Tel Aviv).

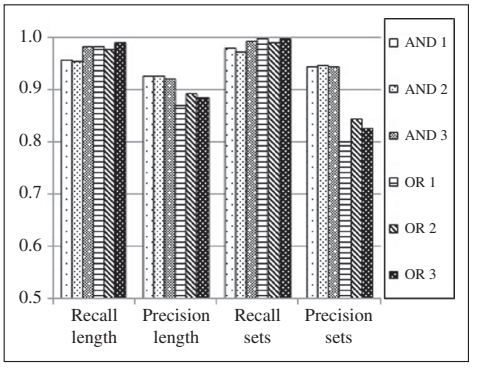

Figure 19. Recall and precision, considering just pairs (Haifa).

7.3. Result analysis

In this section, we first discuss the results of the basic algorithms, and then we explain the effect of the modifications for dealing with length anomalies.

7.3.1. Results of the basic algorithm

We first discuss how the results indicate some characteristics of the different datasets. As expected, in all the tests, the number of nodes satisfying Condition II (counting nodes

Table 13. Recall and precision of the result when dealing with anomalies (Haifa).

	Recall length	Precision length	Recall sets	Precision sets
AND 1	0.93	0.95	0.94	0.95
OR 1	0.92	0.86	0.93	0.86
AND 2	0.92	0.94	0.93	0.94
OR 2	0.91	0.89	0.92	0.89
AND 3	0.94	0.96	0.94	0.96
OR 3	0.92	0.88	0.92	0.88

Table 14. Recall and precision considering just pairs, when dealing with anomalies (Haifa).

	Recall length	Precision length	Recall sets	Precision sets
AND 1	0.95	0.96	0.93	0.96
OR 1	0.97	0.90	0.95	0.81
AND 2	0.94	0.96	0.93	0.97
OR 2	0.97	0.93	0.94	0.86
AND 3	0.97	0.96	0.95	0.96
OR 3	0.98	0.92	0.95	0.84

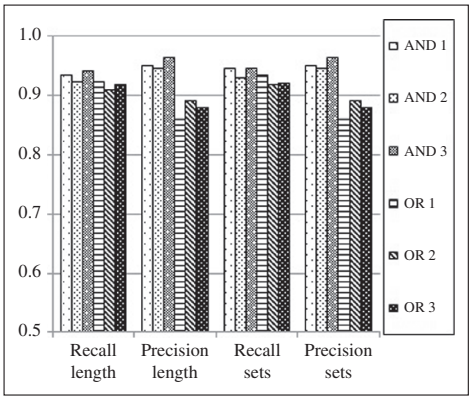

Figure 20. Recall and precision when dealing with anomalies (Haifa).

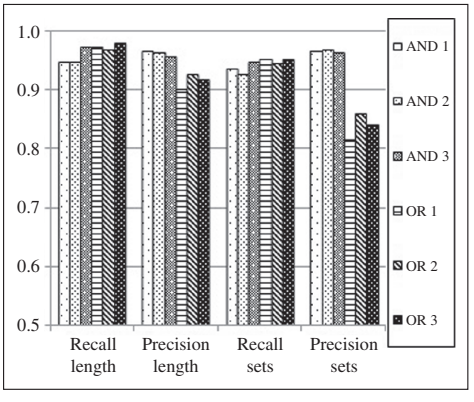

Figure 21. Recall and precision when dealing with anomalies, considering only pairs (Haifa).

where the degree is either equal to 1 or greater than 2) is larger than the number of nodes satisfying Condition I (counting only nodes of degree greater than 2) and, similarly, the number of nodes satisfying Condition III (counting nodes of any degree) is larger than the number of nodes satisfying Condition II. However, in the Tel Aviv datasets and in the Haifa datasets, there is a smaller percentage of nodes satisfying Condition I than in the New York datasets. To see that, compare the results of AND 1 and OR 1 with the results of the other methods (e.g., AND 3 and OR 3) in terms of the number of pairs of nodes that were found in the second step of the algorithm. Over the New York dataset the difference between the methods, when counting the number of pairs, is small (see Table 1), whereas over the Tel Aviv and the Haifa datasets, the difference is large (see Tables 2 and 3). This indicates that the New York road maps are highly connected, while in SOI and MAPA many lines are not connected to the network.

Another characteristic is the similarity between the matched datasets. Tables 4, 7, and 10 show that the percentage of the lines of LION that were matched to lines of CUGIR is greater than the percentage of lines of MAPA that were matched to lines of SOI. This indicates that the similarity of LION to CUGIR is greater than the similarity of MAPA to SOI.

We now discuss factors that affect the quality of the results. In most cases, the recall and precision were higher when we used the length measure than when using the set measure, because many erroneous matches involve relatively short polylines.

In all the tests, the semantics generates more pairs and fewer singletons than the AND semantics. Thus, when counting only pairs, the OR semantics provides a higher recall but a lower precision. When counting also singletons, this is not always the case. The reason for this is that finding more pairs decreases the number of singletons, so the recall may decrease (because correct singletons may not be found) and the precision may increase (because fewer incorrect singletons are present in the result).

Analyzing the effect of using the three conditions is more intricate. It may seem that adding nodes to the first step of the process (by using Condition II and Condition III) would result in finding more pairs of polylines and thus, when counting only pairs, will increase the recall and will decrease the precision. So, Condition I should provide the largest precision and the lowest recall. Similarly, Condition III should provide the lowest precision and the highest recall. This was the case in some of the tests (e.g., when using the New York datasets) but not in all of them.

One reason for having a higher recall when using Condition II than when using Condition III is due to the way the number of pairs are counted in a matching performed by a human. The following example illustrates such a case.

Example 7.1: Consider the polylines in Figure 22. Node 2 and Node b have a degree of two. So, under Condition II both are ignored when computing the matching of the nodes. Under Condition III, these nodes are considered and, assuming that the distance between them is less than β, they are matched. Thus, the output when using Condition II contains three pairs of partial overlap ({1–2, a-b}, {2–3, a-b}, {2–3, b-c}), because Node 2 is an intermediate point in a-b and Node b is an intermediate point in 2–3. When using Condition III, the result contains two pairs of complete overlap ({1–2, a-b}, {2–3, b-c}), because Node 2 and Node b have been matched. When defining the ground truth (the matching generated by a human), we consider this instance as three matches. Hence, in this case, the recall gained when using Condition III is lower than when using Condition II.

Actually, the case in this example occurred rarely in our datasets. In all our tests, there were less than ten such cases.

Adding nodes to the datasets does not necessarily increase the recall. As we match nodes to their nearest neighbors, two matching nodes may no longer be matched when new nodes are added. So, the addition of nodes may discard correct pairs, and thus, may reduce the recall. The following example illustrates this.

Example 7.2: Consider the networks in Figure 23. Node b does not satisfy Condition I. Thus, when Condition I is used, the algorithm matches Node a and Node 1. When either Condition II or Condition III is used, Node b is also considered and, hence, the algorithm matches Node 1 and Node b. So, under the AND semantics, the pair consisting of Node a and Node 1 is not generated.

7.3.2. Dealing with length anomalies

In our tests, dealing with length anomalies improved the test results, over the Haifa datasets, by increasing the precision by almost 2%, helping to gain a precision of 96%–97% (see

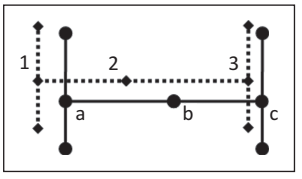

Figure 22. Polyline 1–3 and Polyline a–c can be considered either as two or as three pairs.

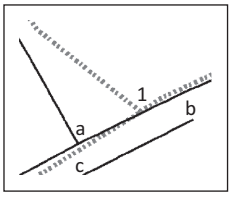

Figure 23. An example where adding nodes may reduce the recall.

Tables 12 and 14). We could not find any additional general cases of dealing with anomalies that could further improve the precision of our algorithms. Our algorithms did not reach a precision of 100% mainly due to uncertainty caused by inaccurate locations of nodes. Figures 24 and 25 show two examples of uncertainty.

7.3.3. Using node degrees

In Section 4.6, we presented the optional step of removing, prior to computing the matching of the polylines, pairs of nodes that do not have the same degree. The level of flexibility in the removal can be controlled by the use of a threshold. When the threshold is equal to zero, all the pairs in which nodes do not have the same degree are discarded. When the threshold is some $k \geq 1$, we do not discard pairs of nodes such that the difference between their degrees is less than or equal to k, but we do discard pairs such that the difference between their degrees exceeds k. In general, an increase in the size of the threshold yields an increase in the flexibility, namely, as the size of the threshold grows, there tends to be fewer removals of pairs.

We ran experiments with different values of the threshold. The effect of changing the threshold on the integration is shown for the New York datasets in Figures 26 and 27, for the Tel Aviv datasets in Figures 28 and 29, and for the Haifa datasets in Figures 30 and 31. Note that in order to simplify the presentation of the results, we used in Figures 27, 29, and 31 the harmonic mean[5] of the recall and precision (HRP) instead of showing the recall and precision separately. (In measuring the recall and precision for Figures 27, 29, and 31, we used the basic measure of counting the number of correct pairs and singletons.) Also note that the effect of the removal step on algorithms AND 2 and AND 3 was similar to the effect on AND 1. So, in order to improve the readability of the graphs, the lines of AND 2 and AND 3 are not presented.

In all the tests, increasing the threshold caused an increase in the running time of the algorithms. The reason for this is obvious. Having fewer pairs of nodes in the result of the first stage of the algorithms reduces the work in the stage of matching the polylines. Note that the effect (on the running time) of increasing the threshold is larger under the

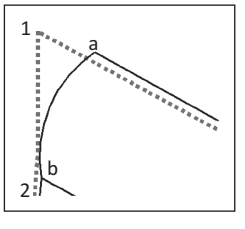

Figure 24. Uncertainty because the location of Junction 1 is imprecise.

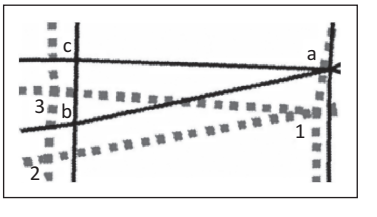

Figure 25. Imprecision leads to matching both Polyline a–b and Polyline a–c to Polyline 1–3.

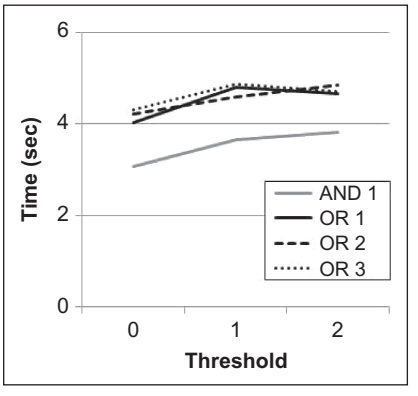

Figure 26. Time (sec) versus threshold (New York).

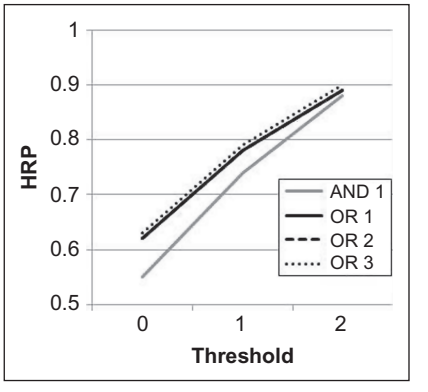

Figure 27. HRP versus threshold (New York).

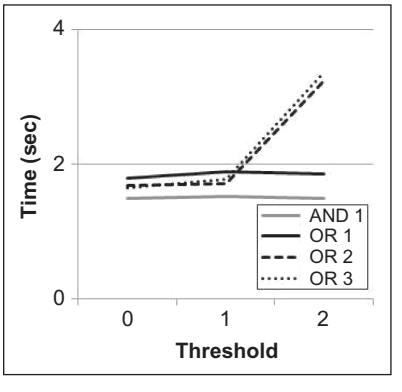

Figure 28. Time (sec) versus threshold (Tel Aviv).

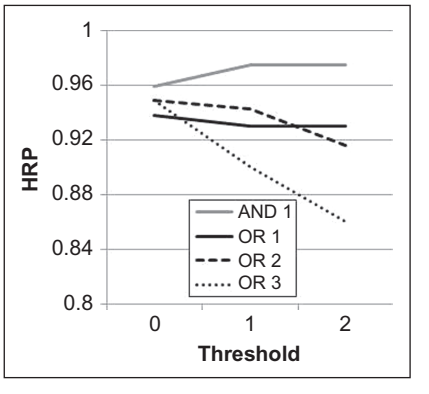

Figure 29. HRP versus threshold (Tel Aviv).

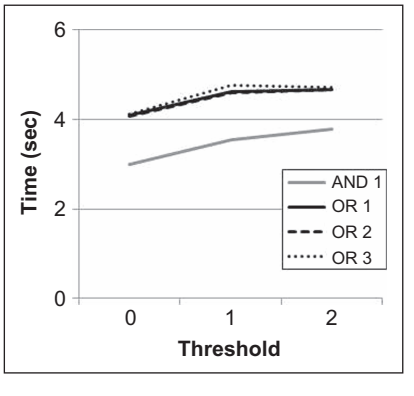

Figure 30. Time (sec) versus threshold (Haifa).

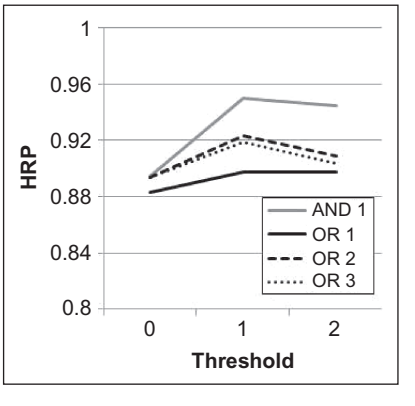

Figure 31. HRP versus threshold (Haifa).

AND-semantics than under the AND-semantics, because more pairs are produced under the OR-semantics, especially in OR 2 and OR 3, than under the AND-semantics.

For the New York datasets, increasing the threshold always increased the HRP. This is because these datasets are accurate but incomplete. Consequently, the matching produced many correct pairs, such that their nodes have different degrees. Removing those pairs reduced the recall and, thus, reduced the HRP.

For the Tel Aviv and the Haifa datasets, lowering the threshold from 2 to 1 improved the quality of the matching (or left it almost unchanged). Evidently, this lowering of the threshold removed more pairs under the OR semantics than under the AND semantics. Thus, the removal has a greater effect on algorithms OR 2 and OR 3 than on AND 1. In the tests over the Haifa datasets, a threshold of 0 gave the worst quality (i.e., lowest HRP) in all the algorithms. This is because over these datasets, the algorithms produced many correct pairs that have a degree difference of at least one.

7.4. *Efficiency of AproxMatching*$_{(\chi,\diamond)}$

The execution times of our algorithms on the datasets that were described earlier are shown in Table 15. For each city, the first column presents the time of the first step – retrieving the topological nodes and creating the data structures L_{point} and L_{polyline} that were described in Section 4.4. The second column shows the time of the second step, that is, matching the topological nodes. The third column presents the time that was required for actually matching the polylines. The experiments were conducted on a PC having a Core 2 Duo processor of 2.13 GHz (E6400) and 2 GB of main memory. Note that for datasets with several hundreds of polylines, the algorithms and especially AND3 complete their task within a few seconds.

To test the efficiency over large datasets, we measured the running times of our algorithms over the entire New York datasets that were described in Section 7.1.1. These datasets contain 25,590 polylines (7326 in CUGIR and 18264 in LION). The computation times of the algorithms over these datasets are shown in Table 16.

In general, the efficiency of the node-matching step can be improved by using an index that provides for each node, the nearest neighbor of that node. We did not use such an index in the experiments that are presented in Table 15, because in these experiments, the running time of the second step had only a small effect on the total computation time. However, when the number of nodes is large, the node-matching step requires a large percentage of the total computation time, in comparison to the case where the number of endpoints is

Table 15. Running times, in seconds, of the tests on New York, Tel Aviv, and Haifa.

	Tel Aviv (569 polylines)			Haifa (1364 polylines)			New York (1349 polylines)		
Stage	1	2	3	1	2	3	1	2	3
AND 1	0.2	0.03	1.73	0.09	0.14	6.25	0.16	0.25	4.98
OR 1			1.96			7.07			4.98
AND 2		0.17	1.64		0.70	5.80		0.25	4.98
OR 2			5.11			10.75			4.70
AND 3		0.20	1.49		0.75	5.90		0.34	3.20
OR 3			5.14			11.75			5.14

The results are presented in the order of the size of the dataset: first, Tel Aviv, then Haifa, and finally, New York.

Table 16. Running times, in seconds, over datasets with 25,590 polylines.

	Stage 1	Stage 2		Stage 3
		Without index	With index	
AND 1	34	54	0.06	76
OR 1			0.14	115
AND 2		57	0.06	77
OR 2			0.16	115
AND 3		76	0.06	79
OR 3			0.22	122

small. Hence, in the experiments over large datasets, we used a hash-based index to improve the efficiency of the node-matching step. In Table 16, we present the running times of the node-matching step with and without the use of an index. It can be seen that using an index considerably improves the efficiency of that step.

7.5. Comparing the efficiency of AproxMatching$_{(\chi,\diamond)}$ with the grid-based methods

In order to show that **AproxMatching**$_{(\chi,\diamond)}$ is efficient for *ad hoc* integration, we now compare it with the two methods that use a grid, namely, the endpoint-buffer method and the polyline-buffer method presented in Section 5. We tested the three methods over the datasets of New York, Tel Aviv, and Haifa. In addition, we consider the following two scenarios that are typical of *ad hoc* integration.

One scenario is when a user provides a small fragment of a road network by taking all the road segments that intersect some bounding box and matches this fragment to a larger map that contains some required information about the roads. The bounding box can be defined by the part of the map that the user sees in some application. In Figure 32, we present such a scenario over the map of Manhattan. A second scenario is when a user

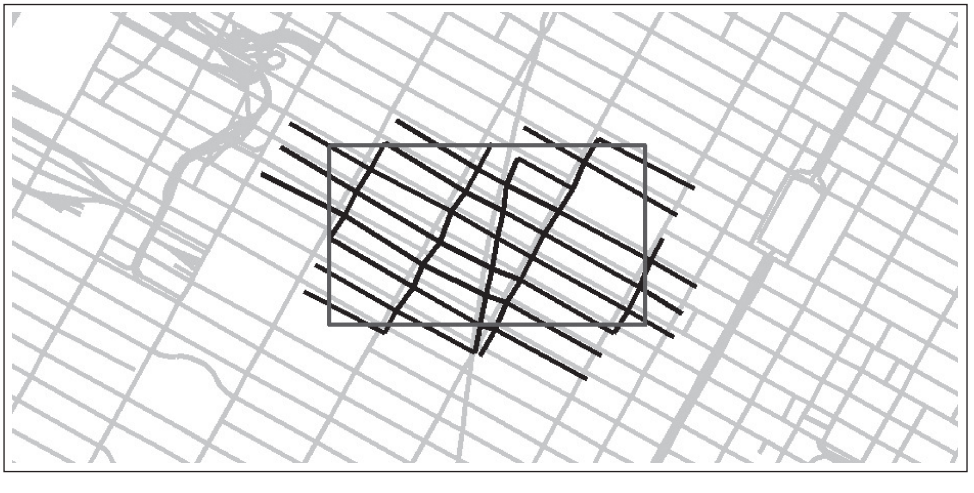

Figure 32. A small fragment of a road network, extracted using a bounding box, is matched to a larger road network. The roads of the small source are depicted in black.

Figure 33. A route is matched to a road network. The road segments of the route are depicted in black.

Table 17. Running times, in seconds, of the three methods for the different integration scenarios.

	Number of polylines					
	Large dataset	Small dataset	Grid creation	Endpoint buffer	Polyline buffer	AND 1
Tel Aviv	404	165	0.047	17.2	28.8	2
Haifa	724	639	0.125	33.6	61.3	6.48
Part of NY	512	815	0.172	45	105.7	5.4
All of NY	18,264	7326	3.67	352.6	853.8	164
BB scenario (Figure 32)	18,264	136	3.67	6.17	12.7	0.7
Route scenario (Figure 33)	18,264	70	3.67	0.75	3.67	0.22

provides a route, and information about the roads of the route should be retrieved from a larger map. Such a scenario is illustrated over the map of New York in Figure 33.

The running times of the methods Endpoint Buffer, Polyline Buffer, and AND 1 are summarized in Table 17. Our experiments show that AND 1 is significantly more efficient than the grid-based methods. The reason for this is that when using a grid, there is a need to apply many expensive geometrical calculations. For instance, computing the grid cells that intersect the buffer of some polyline requires finding for each grid cell whether this cell contains a point of the buffer, that is, finding whether (1) some edge of the cell intersects the buffer or (2) an endpoint of the polyline is inside the cell. Applying such calculations for many grid cells and many polylines is an expensive task. In addition, checking that retrieved polylines of one source indeed correspond to the polylines of the other source also requires expensive geometrical calculations.

The experiments show that the Endpoint-Buffer method is more efficient than the Polyline-Buffer method. The reason for this is that the endpoint buffers cover a smaller

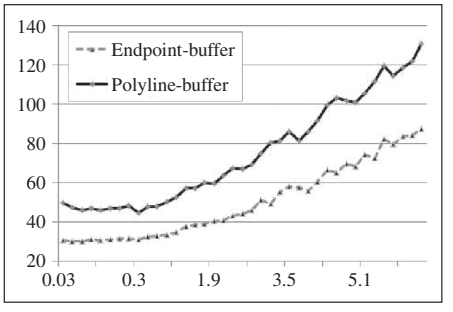

Figure 34. The running time (sec) of the grid-based methods as a function of the size of the grid cells (over the Haifa dataset).

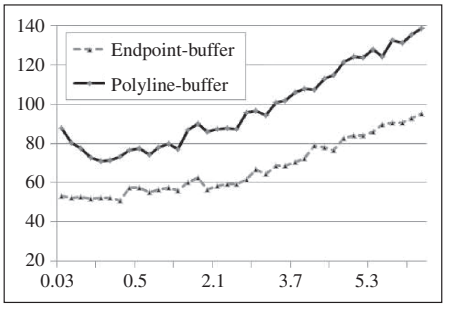

Figure 35. The running time (sec) of the grid-based methods as a function of the size of the grid cells (over the New York dataset).

area than the polyline buffers. Having a smaller part of the map covered leads to retrieving fewer polylines, and this reduces the number of tests that check correspondence of two polylines, thereby decreasing the number of complex geometrical calculations.

When applying the grid-based methods, the size of the grid cells affects the efficiency of the methods. On the one hand, it is desirable to use small grid cells for the following reason. When a group of cells cover a buffer, some extraneous area outside of the buffer is also included in those cells. The smaller the cells, the smaller the extraneous area and hence fewer unnecessary correspondence tests are done. On the other hand, using small grid cells incurs a bigger overhead and increases the number of calculations for finding the cells that intersect a given buffer.

The effect of the size of the grid cell on the efficiency of the methods is illustrated in Figure 34 (over the Haifa dataset) and Figure 35 (over part of the New York dataset). These figures present the running times of the methods as a function of the size of the grid cell. The running times are in seconds. The size of the grid cell is given as a fraction of the mutual error bound β. For instance, 0.5 means that the length of the side of the cell is 0.5β. In both figures, the solid line corresponds to the polyline buffer and the dashed line corresponds to the endpoint buffer. Additional experiments on the other datasets gave similar results. The figures show that the best performance (i.e., the minimum point in the graphs) is achieved when the side of the grid cell has a size of about β. This size strikes the best balance between the number of grid cells that need to be visited and the number of polylines that are involved in the various tests.

The experiments we conducted show that choosing a proper size of the grid cell can significantly improve the efficiency of the grid-based methods; however, even for an optimal choice of the cell size, the AproxMatching$_{(\chi,\diamond)}$ algorithm is more efficient than the grid-based methods.

7.6. Summary of results

We now summarize the results of our experiments, considering the aims we presented at the beginning of Section 7. For targets (i) and (ii), our tests show that usually the best precision is gained by using the AND-semantics with Condition I. This combination is also the most efficient, in most cases. The best recall is usually achieved by using the AND-semantics with Condition III. An exceptional case in which AND 1 does not provide the highest precision is integration where one of the networks has low connectivity. When the

connectivity is low, many endpoints have a degree of 1, and hence, the algorithm ignores them when it uses Condition I. Consequently, there are many erroneous matchings of nodes causing errors in the matching of the polylines. So, when connectivity is low, AND 2 should be used in order to realize high precision.

Considering target (iii), the experiments show that dealing with length anomalies (short polylines and length differences) improves the accuracy of the result with only a small increase in the running time. Discarding pairs of nodes based on the difference in their degrees increases the efficiency of the algorithms. When the data are incomplete, this type of removal reduces the quality of the result (because incompleteness causes a degree difference in correct pairs). When the data are complete, the removal improves the quality of the result.

8. Conclusions

We have investigated the integration of road networks from two datasets. The novelty of our approach is in developing algorithms that have high recall and precision, even though they only use endpoints of polylines. Our algorithms are highly efficient and can be used in scenarios that require *ad hoc* integration, that is, integration in real time rather than as a preprocessing. In particular, we showed that our approach of matching road segments is more efficient than a matching that is based on a spatial index for retrieving potentially corresponding polylines. In addition to the algorithms and the thorough tests, another important contribution is the framework that includes an analysis of the time and space complexities and measures that indicate the quality of the result of an integration.

Several variants of a basic algorithm were presented. Each variation uses one of two semantics for node matching (namely, AND and OR) and one out of three node conditions. Each variation performs better under different circumstances (e.g., whether some of the roads are isolated from the main network or not) and user requirements (e.g., whether the user wants higher recall or higher precision, and whether the result should include only pairs or also singletons).

We tested the different variants of our algorithm on three different integration scenarios. We conducted the tests over four different data sources, from different vendors, consisting of the road networks of the cities of New York, Tel Aviv, and Haifa. Our tests show that when one of the networks has low connectivity, usually AND 2 provides the highest precision. In other cases, the best precision and the highest efficiency are realized by using AND 1. The best recall is typically achieved by using the OR 3 semantics. The experiments also show that our algorithm, which exploits the network for the retrieval of edges, is more efficient than a direct retrieval of edges using a grid index and, hence, is more suitable for *ad hoc* integration than the use of a grid index.

Several interesting problems remain for future work. One is to deal with other types of datasets, such as water and electricity networks. A second problem is how to present the result of the integration graphically as a new map. A third problem is how to use integration of roads to improve the quality and the efficiency of integration of maps that contain both roads and other types of data. A fourth problem is dealing with cases where the coordinate systems of the datasets are not known, for example, when road vector data are extracted from an old raster map.

Acknowledgements

The work of Yaron Kanza and Yehoshua Sagiv was partially supported by The Israeli Ministry of Science and Technology (Grant 3-6472).

Notes

1. DCPNY (LION): `http://www.ci.nyc.ny.us/html/dcp/home.html`
2. CUGIR: `http://cugir.mannlib.cornell.edu`
3. SOI: `http://www.mapi.gov.il/`
4. MAPA: `http://www.mapa.co.il/`
5. HRP $= \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precison}}$

References

Beeri, C., *et al*., 2004. Object fusion in geographic information systems. *In*: Proceedings of the thirtieth international conference on very large data bases, Vol. 30. New York: ACM, pp. 816–827.

Beeri, C., *et al*., 2005. Finding corresponding objects when integrating several geo-spatial datasets. *In*: Proceedings of the 13th annual ACM international workshop on geographic information systems. New York: ACM, pp. 87–96.

Boucelma, O., Essid, M., and Lacroix, Z., 2002. AWFS-based mediation system for GIS interoperability. *In*: Proceedings of the 10th ACM international symposium on advances in geographic information systems. New York: ACM, pp. 23–28.

Chen, C., Knoblock, C.A., and Shahabi, C., 2006. Automatically conflating road vector data with orthoimagery. *GeoInformatica*, 10, 495–530.

Cobb, M.A., *et al*., 1998. A rule-based approach for conflation of attribute vector data. *GeoInformatica*, 2, 7–35.

Cormen, T.H., *et al*., 2009, *Introduction to algorithms*. 3rd ed. The MIT Press.

Devogele, T., Parent, C., and Spaccapietra, S., 1998. On spatial database integration. *International Journal of Geographical Information Science (IJGIS), Special Issue on System Integration*, 12, 335–352.

Doytsher, Y. and Filin, S., 2000. The detection of corresponding objects in a linear-based map conflation. *Surveying and Land Information Systems*, 60, 117–128.

Doytsher, Y., Filin, S., and Ezra, E., 2001, Transformation of datasets in a linear-based map conflation framework. *Surveying and Land Information Systems*, 61, 159–169.

Filin, S. and Doytsher, Y., 1999. A linear mapping approach to map conflation: Matching of polylines. *Surveying and Land Information Systems*, 59, 107–114.

Fonseca, F.T. and Egenhofer, M.J., 1999. Ontology-driven geographic information systems. *In*: Proceedings of the 7th ACM international symposium on advances in geographic information systems. New York: ACM, pp. 14–19.

Fonseca, F.T., Egenhofer, M.J., and Agouris, P., 2002. Using ontologies for integrated geographic information systems. *Transactions in GIS*, 6, 231–257.

Gabay, Y. and Doytsher, Y., 2000. An approach to matching lines in partly similar engineering maps. *Geomatica*, 54, 297–310.

Hangouet, J.F., 1995. Computation of the Hausdorff distance between plane vector polylines. *In*: Twelfth international symposium on computer-assisted cartography (AUTO-CARTO), Charlotte, NC, 27 February–2 March 1995, 1–10.

Haunert, J.H., 2005. Link based conflation of geographic datasets. *In*: Proceedings of the 8th ICA workshop on generalisation and multiple representation. Elsevier.

Laurini, R., Yetongnon, K., and Benslimane, D., 2002. GIS interoperability, from problems to solutions, In EOLSS, The Encyclopedia of Life Support Systems.

Li, L. and Goodchild, M.F., 2010. Automatically and accurately matching objects in geospatial datasets. *In*: Proceedings of the joint international conference on theory, data handling and modelling in GeoSpatial Information Science. Hong Kong: ISPRS Technical Commission II.

Lowell, K. and Jaton, A., 1999. *Spatial accuracy assessment: Land information uncertainty in natural resources*. Chelsea, MI: CRC Press.

Mikhail, E.M. and Ackermann, F.E., 1982. *Observations and least squares*. Lanham, MD: University Press of America.

Mustière, S. and Devogele, T., 2008. Matching networks with different levels of detail. *GeoInformatica*, 12, 435–453.

Noronha, V., 2000. Towards ITS map database interoperability—database error and rectification. *GeoInformatica*, 5, 345–373.

Parent, C., Spaccapietra, S., and Zimányi, E., 2006. The MurMur project: Modeling and querying multi-representation spatio-temporal databases. *Information Systems*, 31, 733–769.

Rosen, B. and Saalfeld, A., 1985. Match criteria for automatic alignment. *In*: Proceedings of the 7th international symposium on computer-assisted cartography (Auto-Carto). U.S. Bureau of the Census, pp. 1–20.

Saalfeld, A., 1988. Conflation-automated map compilation. *International Journal of Geographical Information Systems* (*IJGIS*), 2, 217–228.

Safra, E., *et al*., 2006. Integrating data from maps on the world-wide web. *In*: Proceedings of the 6th international symposium on web and wireless geographical information systems (W2GIS). Berlin: Springer, pp. 180–191.

Safra, E., *et al*., 2010. Location-based algorithms for finding sets of corresponding objects over several geo-spatial data sets. *International Journal of Geographical Information Science* (*IJGIS*), 24, 69–106.

Scholl, M.O., Rigaux, P., and Voisard, A., 2002. Spatial databases with application to GIS. San Francisco, CA: Morgan Kaufmann Publishers Inc.

Sester, M., Anders, K.H., and Walter, V., 1998. Linking objects of different spatial data sets by integration and aggregation. *GeoInformatica*, 2, 335–358.

Tong, X., Shi, W.Z., and Deng, S., 2009. A probability-based multi-measure feature matching method in map conflation. *International Journal of Remote Sensing*, 30, 5453–5472.

Uitermark, H., *et al*., 1999. Ontology-based geographic data set integration. *In*: Proceedings of the international workshop on spatio-temporal database management (STDBM). Edinburgh: Elsevier, pp. 60–79.

Volz, S. and Walter, V., 2004. Linking different geospatial databases by explicit relations. *In*: Proceedings of the 10th ISPRS Congress, Vol. 35. ISPRS, pp. 152–157.

Walter, V. and Fritsch, D., 1999. Matching spatial data sets: A statistical approach. *International Journal of Geographical Information Science (IJGIS)*, 13, 445–473.

Ware, J.M. and Jones, C.B., 1998. Matching and aligning features in overlayed coverages. *In*: Proceedings of the 6th ACM international symposium on advances in geographic information systems (ACM-GIS), November 6–7, 1998, Washington, DC. New York: ACM, pp. 28–33.

Wiederhold, G., 1992. Mediators in the architecture of future information systems. *Computer*, 25, 38–49.

Wiederhold, G., 1999. Mediation to deal with heterogeneous data sources. *In*: Proceedings of the interoperating geographic information systems. Springer, pp. 1–16.

Xiong, D.M. and Sperling, J., 2004. Semiautomated matching for network database integration. *ISPRS Journal of Photogrammetry & Remote Sensing*, 59, 35–46.