Efficient Integration of Road Maps

Eliyahu Safra
Technion
Haifa, Israel
safra@technion.ac.il

Yaron Kanza*
University of Toronto
Toronto, Canada
yaron@cs.toronto.edu

Yehoshua Sagiv†
The Hebrew University
Jerusalem, Israel
sagiv@cs.huji.aci.il

Yerach Doytsher
Technion
Haifa, Israel
doytsher@technion.ac.il

ABSTRACT

Integration of two road maps is finding a matching between pairs of objects that represent, in the maps, the same real-world road. Several algorithms were proposed in the past for road-map integration; however, these algorithms are not efficient and some of them even require human feedback. Thus, they are not suitable for many important applications (*e.g.*, Web services) where efficiency, in terms of both time and space, is crucial. This paper presents two efficient algorithms for integrating 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. Experiments on real-world data are given, showing that our approach of integration based on matching merely endpoints is efficient and accurate (that is, it provides high recall and precision).

Categories and Subject Descriptors:
H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms: Algorithms, Experimentation

Keywords: Road map, spatial network, integration, matching polylines, location-based join, corresponding objects

1. INTRODUCTION

Digital road maps represent electronically a network of roads. 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, etcetera. Such applications may need to use both spatial and non-spatial properties of roads. Integration of two road maps makes it possible for the applications to use properties of a road that are represented in only one of the maps, and at the same time use properties that are represented only in the other 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 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 cases, the efficiency of the integration process is crucial. One case is of applications provided by a Web server that should handle many users concurrently. A second example is of applications that run in devices with a limited processing power, *e.g.,* a hand-held device such as a PDA. In these cases, when called by a person walking or driving a car, the applications should provide the answer immediately (say, within a few seconds).

Several methods for integrating road maps were proposed in the past [4, 5, 9]; however, these methods are not efficient. They are designed for finding answers that are as accurate as possible without taking efficiency into consideration. Hence, these methods require a long computation time and they are not suitable for scenarios where efficiency is crucial.

In this paper, we consider maps in which roads are represented by polygonal lines (polylines). 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 merely on locations of endpoints of polylines rather than trying to match whole lines. There are two important advantages to our approach. First, integration can be done efficiently. Secondly, 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. Differently from other properties, locations always exists 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, we do not even use the topology of the road network. This is because using the topology may increase the complexity of the computation. Furthermore, using 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.

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

*This author was supported by an NSERC grant.
†This author was supported by The Israel Science Foundation (Grants 96/01 and 893/05).

ACM-GIS'06, November 10–11, 2006, Arlington, Virginia, USA.

locations. Second, endpoints may be chosen differently in the two maps and, hence, an endpoint in one map may be located in the middle of a line 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, so, the two maps can use different approximations. Third, information might be incomplete so that a road or a segment of a road, in one map, may not appear in the other map, and vice versa.

In this paper, we introduce a method for integrating two digital road maps that matches polylines merely according to the locations of their endpoints. The method is based on finding a partial matching between the endpoints of polylines in the two sources. We discuss two semantics for matching endpoints, namely, the AND *semantics* and the OR *semantics*. Under the AND semantics, two endpoints are matched if each one is the nearest neighbors of the other. Under the OR semantics, two endpoints are matched if at least one of the points is the nearest neighbor of the other.

In order to show the efficiency and the effectiveness of our techniques, we conducted experiments on real-world data. In the tests, we compared the AND and the OR semantics. Also, we investigated 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, *i.e.*, they provide high recall and high precision. The tests also show that the best performance, in terms of both efficiency and accuracy, is for the AND semantics when using only endpoints that are intersections of three or more roads.

2. FRAMEWORK

In this section we present our framework. We provide formal definitions to the notion of a road map in a geo-spatial database. Also, we discuss the notation of a matching algorithm and we describe the result of such an algorithm.

2.1 Road Map

A *road map* 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 another road. 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 *polygonal line* (abrv. *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 non-consecutive 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 geo-spatial 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, *e.g.,* 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 non-spatial attributes. Spatial attributes describe the location, length, shape and topology of a road. Examples of non-spatial attributes are road number, traffic direction, number of lanes, speed limit, etc.

2.2 Matching Corresponding Objects

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. 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 approximated matching is computed according to the properties of the spatial objects. In our approach, we compute matchings based on the location of objects. This is because locations are always available for spatial objects. Also, 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. Secondly, 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 is based merely on the location of the endpoints of the polylines.

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 [1, 2] can compute an approximate matching of the nodes, based on their point locations. In principle, the join sets computed in this 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.

2.3 Error Bound

In an integration process, the accuracy of the datasets must be taken into account. Object locations are never completely accurate. The accuracy of locations is influenced by

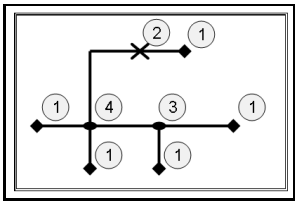

Figure 1: Nodes and their degree

several factors, such as the techniques used to measure locations, 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 paper, 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 $\beta = \sqrt{{m_1}^2 + {m_2}^2}$. 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 our algorithms, the standard deviation σ of each dataset is provided. The error bound β is computed and pairs of objects are candidates for being corresponding objects only if the distance between them does not exceed β.

3. COMPUTING A MATCHING

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 $\textbf{AproxMatching}_{(\chi,\square)}$ is presented in Figure 2. Under the AND-semantics, $\square$ should be replaced with the `and` logical operator, *i.e.,* conjunction; and under the OR-semantics, $\square$ should be replaced with the `or` logical operator, *i.e.,* disjunction. Variations of the basic algorithm are also obtained by considering as nodes only endpoints that satisfy a given condition χ. Possible conditions and their effect are discussed in the following section.

3.1 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 intersections of at least three roads, all intersections of at least three roads 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.

1. **Condition I:** The node degree is greater than 2.
2. **Condition II:** The node degree is different from 2, *i.e.*, is either equal to 1 or greater than 2.
3. **Condition III:** The node degree is any number.

EXAMPLE 3.1. In Figure 1, a network of roads is depicted. The degree of each node is written next to the node. All the nodes in this network, except for the one marked by an $\times$ (*i.e.,* the one whose degree is 2) satisfy Condition II.

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

Input: Two road maps M_1 and M_2
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 $\square \in \{\text{AND}, \text{OR}\}$

1: **for** $i = 1, 2$ **do**
2: let S_i be all the pairs (n, l), such that n is an endpoint of 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 lines for which n is an endpoint
6: let N_i be the set of nodes in S_i that satisfy χ
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 $N_1) \square (n_2$ 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
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 2: Computing an approximate matching of polylines

The step of finding the relevant nodes, of each dataset M_i, is presented in Lines 1–6 of the algorithm of Figure 2. 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 sort 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.

3.2 Matching Nodes

In the second step, the algorithms compute an approximate matching μ_n over the nodes of the sets N_1 and N_2 obtained in the first step. The approximation depends on

the chosen semantics. Under the AND-semantics, *i.e.,* when $\square$ 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* [1, 2]. Note that under the AND-semantics, each node appears in exactly one pair of corresponding objects.

Under the OR-semantics, *i.e.,* when $\square$ 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 in 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 2.3) 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.

3.3 Matching Polylines

In the third and final step, the algorithms compute 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 2). Then, singletons are created from all the remaining polylines (Lines 12–14 of Figure 2).

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.

1. *Complete overlap*: We say that there is a complete overlap for l_1 and l_2 if they have two pairs of corresponding endpoints, *e.g.,* both $\{n_1, n_2\}$ and $\{n_1', n_2'\}$ are corresponding nodes.

2. *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, *e.g.,* $\{n_1, n_2\}$ are corresponding nodes and n_2' is an intermediate point in l_1.

3. *Containment*: We say that l_1 contains l_2 when both endpoints of l_2 are intermediate points of l_1.

4. *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, *e.g.,* the node n_1 is an intermediate point in l_2 and the node n_2' is an intermediate point in l_1.

EXAMPLE 3.2. Figure 4 shows the four relationships between corresponding polylines.

We denote by $in(n, l)$ a 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 2.3. 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. When l is made of more than one segment, first, the nearest point to n in each one of the segments can be computed by applying an orthogonal projection of n on these segments. Then, n' is the point with the shortest distance from n among the points found by the orthogonal projections.

Match-Lines(S_1, S_2, μ_n)

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

```
 1: μ_l ← ∅, I ← ∅, V ← ∅
 2: for each pair {n1, n2} ∈ μ_n do
 3:    for each (n1, l1) ∈ S1 and (n2, l2) ∈ S2 do
 4:       if exist (n1', l1) ∈ S1 and (n2', l2) ∈ S2 s.t.
          n1' ≠ n1 and n2' ≠ n2 and {n1', n2'} ∈ μ_n then
 5:          add {l1, l2} to μ_l
 6:       if exists (n1', l1) ∈ S1 s.t. in(n1', l2) then
 7:          add {l1, l2} to μ_l
 8:          add (n1', l2, 1) to I
 9:       if exists (n2', l2) ∈ S2 s.t. in(n2', l1) then
10:          add {l1, l2} to μ_l
11:          add (n2', l1, 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') ∈ S_i and (n', l') ∈ S_i s.t.
       in(n', l) do
16:       add {l, l'} to μ_l
17:       if (n', l, i) is not in V then
18:          add (n', l, i) to I
19:    for each (n, l') ∈ S_i and (n', l) ∈ S_j s.t.
       in(n', l') do
20:       add {l, l'} to μ_l
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 3: Finding corresponding polylines

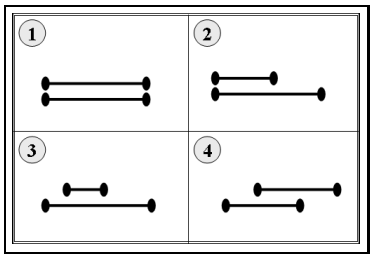

Figure 4: The four relationships between corresponding polylines

The matching of the polylines according to the above four relationships is computed by the method **Match-Lines** presented in Figure 3. The method receives polylines, their endpoints (the sets S_1 and S_2) and a correspondence relationship μ_n for the nodes. It returns a set μ_i consisting of pairs of corresponding polylines.

The method uses two supporting data structures. A stack I is used for storing triplets of a polyline, a node whose location is an intermediate point in the polyline and 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 lines that have a complete overlap or an extension relationship. In the case of a complete overlap, the pair of lines is simply added to μ_i (Lines 4–5). In the case of an extension, the pair of lines is added to μ_l and, in addition, the triplet for the line and the node, where the node is an intermediate point in the line, 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 of the algorithm, when Line 12 is reached, I already contains the intermediate points that were discovered during the search for lines 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 were already 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 lines that have a containment or a partial-overlap relationship. This is because in the traversal over the nodes, the algorithm will not visit intermediate nodes that are isolated, that is, intermediate nodes that are not connected by an edge to a visited node. This was done on purpose, since our goal is to provide an approximate matching while keeping the algorithm efficient.

3.4 Time and Space Complexity

In this section, we analyze the time and space complexities of the method $\mathbf{AproxMatching}_{(\chi,\square)}$. 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 $O(\min\{k_1, k_2\} + S_{nn}(k_1) + S_{nn}(k_2))$, since 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 $O(k_1 + k_2 + S_{nn}(k_1) + S_{nn}(k_2))$, since 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 objects. The number of corresponding objects 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, no more than d edge tests are conducted with respect to each popped node. Retrieving the edges 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 3.3. *Let M_1 and M_2 be road maps containing k_1 and k_2 polylines, respectively, and suppose that $k1 \geq k_2$.*

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

4. QUALITY OF 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. C denotes the set of correct join sets appearing in the result. A is the set of all the correct join sets. 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 the 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 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. Then, in the basic length-based measure, the recall r and the precision p are

$$r = \frac{\sum_{s \in C} length(s)}{\sum_{s \in A} length(s)} \qquad p = \frac{\sum_{s \in C} length(s)}{\sum_{s \in R} length(s)}. \tag{1}$$

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

By and large, methods that provide high recall and precision according to length-based measures are good for integrating road maps of rural areas, *i.e.,* 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, *i.e.,* 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, we considered the set A of all the correct join sets to be the join sets found by the expert, and we computed our measures with respect to this set.

5. EXPERIMENTS

In this section, we describe our experiments for determining the efficiency and accuracy of the six variants of the basic algorithm **AproxMatching**$_{(\chi,\square)}$ of Figure 2. We use AND and OR to denote the two semantics of Section 3.2 for matching nodes. The numerals 1, 2 and 3 indicate the three conditions of Section 3.1 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.

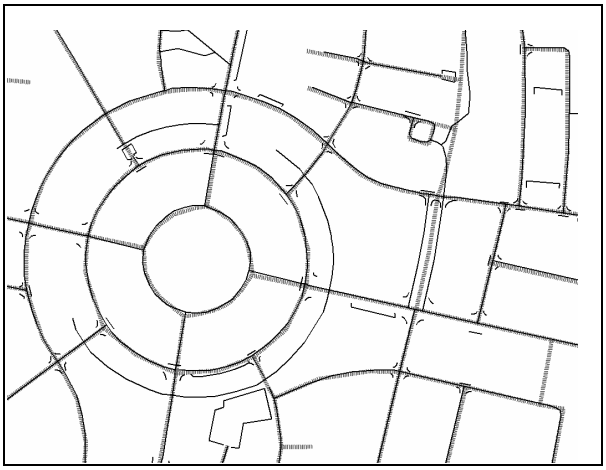

Figure 5: A visual view of the two datasets: SOI (solid lines) and MAPA (dashed lines)

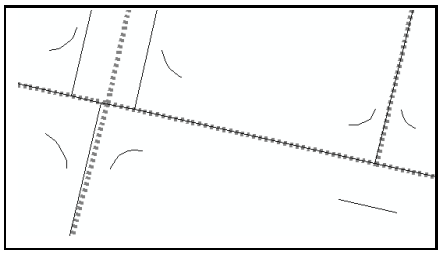

Figure 6: A visual view of the vicinity of two junctions (SOI depicted by solid lines and MAPA depicted by dashed lines)

The experiments were aimed at answering the following questions. First, which of the three conditions is best for selecting nodes? Second, which of the two semantics gives better results? We considered both questions with respect to the different ways of measuring the quality of the result (*i.e.*, length vs. sets).

5.1 Input Datasets

For the experiments, we used two real-world datasets that were collected by different organizations, at different times and with different collection methods (see Figure 5). The first dataset, denoted as SOI, was collected by the Survey of Israel. This dataset was extracted from aerial photographs at the scale of 1:40,000 (equivalent to digital maps at the scale of 1:5,000–1:10,000). The second dataset, called MAPA, was extracted directly from a digital map, at the scale of 1:25,000, by the Mapa Corp. The two datasets describe roads in Tel-Aviv. The dataset of SOI contains 420 polylines. The dataset of MAPA contains 167 polylines.

Part of the test area can be seen in Figure 5. The test area contains regions with different characteristics: residential versus commercial districts, straight versus curved lines and ordered versus disordered areas. Although the two maps have a similar scale, there is a large difference in how they describe the same roads. SOI is oriented to the generation of maps and other geometric measurements, while MAPA is used mainly for road navigation.

	# of nodes in SOI	# of nodes in MAPA	# of pairs
AND 1	107	84	66
OR 1			72
AND 2	532	113	83
OR 2			150
AND 3	554	126	91
OR 3			168

Table 1: The number of nodes and the number of pairs of nodes

	Pairs in the Result	Singletons in the Result	Correct Pairs	Correct Singletons
AND 1	170	111	170	96
OR 1	177	107	173	93
AND 2	168	112	167	92
OR 2	217	105	171	90
AND 3	174	104	170	86
OR 3	231	92	177	83

Table 2: The number of join sets in the result and the number of correct join sets

	Recall length	Precision length	Recall sets	Precision sets
AND 1	0.94	0.92	0.91	0.95
OR 1	0.94	0.93	0.91	0.94
AND 2	0.93	0.93	0.89	0.93
OR 2	0.92	0.92	0.89	0.81
AND 3	0.92	0.92	0.88	0.92
OR 3	0.92	0.93	0.89	0.80

Table 3: Recall and precision of the result

Figure 6 shows a small vicinity with two junctions. It can be seen that SOI has some isolated polylines while in MAPA all the polylines form a connected network.

Table 1 gives, for each of the six variants, the number of nodes that were created from each dataset (in the first step of the algorithm) and the number of matching pairs of nodes that were found in the second step. We manually determined the perfect matching and it consisted of 292 join sets: 195 pairs and 97 singletons having total lengths of 13,900 and 7,937 meters, respectively.

Determining the correct joins sets was not always straightforward. For example, in Figure 6, it is not clear if the short roads from SOI should be matched to zero, one or two objects of MAPA. In such cases, the objects were not included in any correct join set and, in the result, join sets containing these objects were ignored.

5.2 Results

Table 2 shows, for each experiment, the numbers of pairs and singletons that were produced and the numbers of the correct join sets. Table 3 and Figure 7 show the recall and precision of each algorithm, using the two methods for measuring the quality of the result. Table 4 and Figure 8 show the same, but only for pairs; that is, the perfect matching has only the correct pairs and, in the result of each algorithm, the singletons are ignored. Note that in Figures 7 and 8, bars with horizontal lines are for experiments that use Condition I, bars with dots are for Condition II and bars with vertical lines are for Condition III.

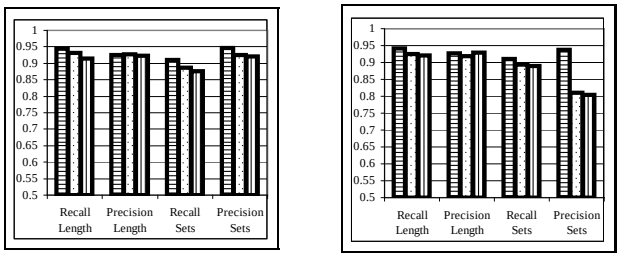

Figure 7: Recall and precision for the and semantics (left) and for the or semantics (right)

	Recall length	Precision length	Recall sets	Precision sets
AND 1	0.92	1.00	0.87	1.00
OR 1	0.93	0.99	0.89	0.98
AND 2	0.92	1.00	0.86	1.00
OR 2	0.93	0.95	0.88	0.79
AND 3	0.92	0.98	0.87	0.98
OR 3	0.95	0.94	0.91	0.77

Table 4: Recall and precision of pairs in the result

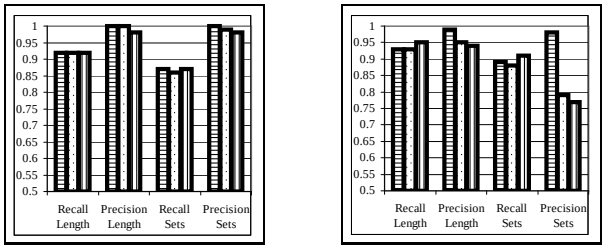

Figure 8: Recall and precision, when only the matching of pairs is measured, for the and semantics (left) and for the or semantics (right)

The experiments show that the algorithm "AND 1" has the best performance in terms of both efficiency and accuracy. Note, however, that the algorithm "OR 3" has a higher recall (when only pairs are considered) at the cost of a lower precision. In the maps we used, the polylines form an almost completely connected network of roads. So, using Condition II did not increase the recall. However, when some polylines are isolated, then all their nodes have a degree of at most 2. Thus, either Condition II or Condition III must be used in order to match these polylines.

In most cases, the recall and precision were higher according to the length measure, because many erroneous matches involve relatively short objects.

5.3 Running Times

The execution time (in seconds) of each algorithm is shown in Table 5. The first column gives the time of the first step while the second column gives the combined time of the last two steps (recall that the first step is actually a preprocessing that can be done in advance in each source). The experiments were done on a PC equipped with a 3GHz Pentium 4 processor and 512M of RAM.

6. RELATED WORK

In the literature, there are several approaches for line matching. In some approaches, algorithms use mainly ge-

	Preprocessing	Matching
AND 1	0.11	2.83
OR 1		2.94
AND 2	8.73	3.13
OR 2		3.95
AND 3	18.45	3.11
OR 3		4.01

Table 5: Running Times

ometric attributes while in others, algorithms are based on alpha-numeric attributes. One of the first approaches [6, 7] is a two-step iterative process. In the first step, several points are matched using proximity and other attributes. In the second step, a rubber shit is applied in order to correct the locations of the points that were not matched.

In [5], a polyline from one source is augmented with a buffer. When a polyline from the other source is completely contained in that buffer, the two polylines are considered a matching pair. Other matching algorithms that uses buffering appear in [10].

The work of [4] applies a point-matching algorithm as the first step, then uses the topology of polylines to propagate the matching. Finally the average distance between the lines is used to estimate the relative compatibility of the lines.

The above approaches are more conservative than ours when matching two polylines, but they are less efficient since they use all the points along the polylines and do complex geometric computations.

The approach of [3] uses not only the geometric attributes, but also alpha-numeric attributes. This approach applies a preprocessing in order to match similar attributes from the different sources. The preprocessing ends when a set of rules for matching objects is defined. The rules are organized hierarchically from the most significant rule to the least significant rule. The matching is done by applying the rules one by one.

An interesting approach that combines buffering and usage of other attributes appears in [8, 9]. They start with a manual matching of some small test case, which is used for learning the behavior of the datasets. Then, a buffer measure is applied to filter out impossible matches and to create a tree with all the possible matches. Finally, the selection of the best match is done by calculating a function that measures the degree of similarity between the matched area and the test case.

The last two approaches use prior knowledge about the datasets whereas in our approach no prior knowledge is needed. Evidently, these approaches are much less efficient than ours. For example, in [9], the matching of two datasets with 363 and 435 objects took two hours and seven minutes, while in our algorithm, matching of two datasets with 420 and 165 objects took about three seconds.

7. CONCLUSION

We have investigated the integration of road networks from two datasets. The novelty of our approach is in developing an algorithm that has high recall and precision, even though it only uses endpoints of polylines. Our algorithm is much faster than previous ones. In addition to the algorithm itself, 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 the algorithm were presented. One of the variants gives the best recall and precision. Yet, other variants may be used when high recall is needed (at the expense of a somewhat lower precision), or when some of the roads are isolated from the main network.

We tested the different variants of our algorithm on two datasets consisting of the road network of Tel-Aviv. The tests show that, compared to earlier work, the reduction in processing time is huge (seconds instead of hours), while the quality of the results remains almost the same (i.e., a difference of only a few percentage points).

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 handle situations where the input datasets have different scales. The proposed algorithm may cope with such differences up to some limit. However, when the scale difference is large, the result may be confusing. So, additional work is needed.

8. REFERENCES

[1] C. Beeri, Y. Doytsher, Y. Kanza, E. Safra, and Y. Sagiv. Finding corresponding objects when integrating several geo-spatial datasets. In *ACM-GIS*, pages 87–96, 2005.

[2] C. Beeri, Y. Kanza, E. Safra, and Y. Sagiv. Object fusion in geographic information systems. In *VLDB*, pages 816–827, 2004.

[3] M. A. Cobb, M. J. Chung, H. Foley, F. E. Petry, and K. B. Show. A rule-based approach for conflation of attribute vector data. *GioInformatica*, 2(1):7–33, 1998.

[4] Y. Doytsher and S. Filin. The detection of of corresponding objects in a linear-based map conflation. *Surveying and Land Information Systems*, 60(2):117–128, 2000.

[5] Y. Gabay and Y. Doytsher. An approach to matching lines in partly similar engineering maps. *Geomatica*, 54(3):297–310, 2000.

[6] B. Rosen and A. Saalfeld. Match criteria for automatic alignment. In *Proceedings of 7th International Symposium on Computer-Assisted Cartography (Auto-Carto 7)*, pages 1–20, 1985.

[7] A. Saalfeld. Conflation-automated map compilation. *International Journal of Geographical Information Systems*, 2(3):217–228, 1988.

[8] M. Sester, K. H. Anders, and V. Walter. Linking objects of different spatial data sets by integration and aggregation. *GeoInformatica*, 2(4):335–358, 1998.

[9] V. Walter and D. Fritsch. Matching spatial data sets: a statistical approach. *International Journal of Geographical Information Science*, 13(5):445–473, 1999.

[10] J. M. Ware and C. B. Jones. Matching and aligning features in overlayed coverages. In *Proceedings of the 6th ACM International Symposium on Advances in Geographic Information Systems*, pages 28–33, 1998.