Integrating Data from Maps on the World-Wide Web

Eliyahu Safra[1], Yaron Kanza[⋆2], Yehoshua Sagiv[⋆⋆3], and Yerach Doytsher[1]

[1] Department of Transportation and Geo-Information, Technion, Haifa, Israel
{safra, doytsher}@technion.ac.il
[2] Department of Computer Science, University of Toronto, Toronto, Canada
yaron@cs.toronto.edu
[3] School of Engineering and Computer Science, The Hebrew University, Jerusalem, Israel
sagiv@cs.huji.ac.il

Abstract. A substantial amount of data about geographical entities is available on the World-Wide Web, in the form of digital maps. This paper investigates the integration of such data. A three-step integration process is presented. First, geographical objects are retrieved from Maps on the Web. Secondly, pairs of objects that represent the same real-world entity, in different maps, are discovered and the information about them is combined. Finally, selected objects are presented to the user. The proposed process is efficient, accurate (*i.e.*, the discovery of corresponding objects has high recall and precision) and it can be applied to any pair of digital maps, without requiring the existence of specific attributes. For the step of discovering corresponding objects, three new algorithms are presented. These algorithms modify existing methods that use only the locations of geographical objects, so that information additional to locations will be utilized in the process. The three algorithms are compared using experiments on datasets with varying levels of completeness and accuracy. It is shown that when used correctly, additional information can improve the accuracy of location-based methods even when the data is not complete or not entirely accurate.

1 Introduction

Many maps are available on the World-Wide Web, providing information on geographical entities. The information consists of both spatial and non-spatial properties of the entities. Examples of spatial properties are location and shape of an entity. Examples of non-spatial properties are name and address. The goal of integrating two maps is to enable applications and users to easily access the properties that are available in either one of those maps. Another reason for integration is that some geographical entities may appear in only one of the maps. Integration increases the likelihood that for all the relevant entities, in a specified geographical area, objects that represent these entities are presented to the user.

An integration of two maps consists of the following three steps: extracting geographical objects from the maps, discovering pairs of objects that represent the same real-world entity in different sources (such objects are called *corresponding objects*)

⋆ This author was supported by an NSERC grant.
⋆⋆ This author was supported by The Israel Science Foundation (Grant 893/05).

and presenting the result to the user. This paper deals mainly with the second step of discovering corresponding objects. We use the term *matching algorithm* for an algorithm that discovers corresponding objects in two given datasets of geographical objects.

Methods for integrating data from the Web, and especially matching algorithms, should be able to cope with the following characteristics of the Web.

– Data on the Web is heterogeneous. This means that the same piece of information can have different forms in different sources. For example, in different sources, the name of a geographical entity can have different spellings or can be written in different languages. This makes it difficult for integration methods to use properties, such as names, for discovering corresponding objects. Another aspect of heterogeneity is incompleteness. Some attributes may not be available in some sources or not specified for some objects.
– Data may change frequently. For example, maps that contain hotels may also include reviews that are regularly added and updated by people who have stayed in those hotels. In such cases, the integration should be performed in real time, *i.e.*, when the user sends her request for information. Otherwise, the integrated data will not reflect the most recent changes in the sources. Consequently, an integration method for data on the Web must be efficient, especially if the method is used in a Web service that handles many requests concurrently.
– Data on the Web can be incorrect or inaccurate. Hence, on one hand, integration methods should rely mostly on object properties that are relatively accurate. On the other hand, this justifies using, in Web applications, approximation matching algorithms, *i.e.*, *highly* (but not completely) *accurate* algorithms for discovering corresponding objects.

Because of the above reasons, in this paper we consider techniques that start with location-based matching algorithms and improve them. Relying primarily on locations has the following three advantages. First, locations are always available for spatial objects and their degree of accuracy can be determined relatively easily. Hence, location-based matching algorithms can be applied to objects from any pair of maps. Second, location-based methods are suitable for integration of heterogeneous data, since it is easy to compare a pair of locations even when they are stored or measured in different ways. Third, there exist efficient location-based matching algorithms.

Location-based matching algorithms that are both efficient and effective were presented in the past [2, 3]. These algorithms only use locations for finding corresponding objects. Yet, in many cases, the accuracy of the integration can be improved significantly by using attributes of the integrated objects in addition to locations. This is especially important when dealing with data from the Web, where locations may be inaccurate. In this paper, we explain how to use properties of integrated objects to increase the effectiveness of location-based matching algorithms.

The main contributions of this paper are as follows. First, a complete process of integrating data from maps on the Web is presented. This process is efficient and general, in the sense that it can be applied to any pair of maps. Secondly, we show how, in addition to locations, attributes of the objects can be used in the integration process. Specifically, we present three new matching algorithms that use locations as well as additional information. Thirdly, we describe the results of thorough experiments, on datasets with different levels of accuracy and completeness, showing that additional information can improve the results of location-based matching algorithms, when that information is used appropriately.

The structure of the paper is as follows. In Section 2 we present our methods using a real-world example of integrating maps showing hotels in the Soho area of Manhattan, New-York. We present our three new methods in Section 3. In Section 4, we provide the results of experiments we conducted on both real-world data and syntactically generated data. Also, we compare our methods based on the experimental results. Finally, in Section 5, we discuses related work and conclude.

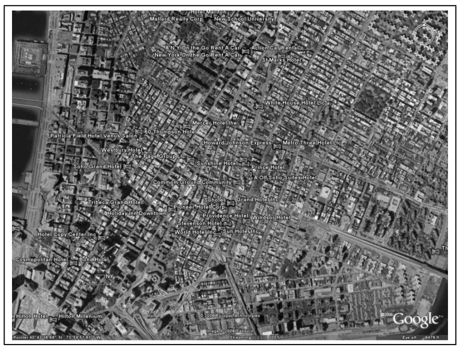
2 The Integration Process

We start by presenting our approach to integration of data from maps on the Web. We do that using an example showing integration of information about hotels in the Soho area of Manhattan, New-York. The data sources we used are Google Earth[4] and Yahoo Maps[5]. Google Earth is a service that provides a raster image of almost any part of earth. On top of the raster image it shows information such as roads, hotels, and restaurants. In our example we are interested in information about hotels. For hotels, Google Earth provides their names. The names are links that lead to additional information, *e.g.,* by following a link the address of the hotel is provided. A result of a search in Google Earth for hotels in Soho is depicted in Fig. 1.

Yahoo Maps provides road maps for some major cities in the world. As in Google Earth, maps include touristic information; however, in Yahoo, hotel names are not presented on the maps. Instead, a hotel is shown using an icon of a yellow square containing a red circle, in the location of the hotel. The name of the hotel and additional information such as the rank (*i.e.*, number of stars) and price are available for one hotel at a time. Two possible reasons for not writing hotel names on the map are *(1)* making the presentation of the map simpler and easier to read (cartographic reasons), and *(2)* restricting the information released per each user request, so that applications will not be able to retrieve all the data from Yahoo to their local database (commercial reasons). A result of a search in Yahoo Maps for hotels in Soho is depicted in Fig. 2.

It may seem a good solution to use, in the hotel scenario, a matching algorithm that consider as corresponding objects, pairs of hotels that have the same name. However, because names of hotels are not presented on maps from Yahoo, a matching based on names is problematic. Two other difficulties in using hotel names in a matching algorithm are the uncertainty in deciding whether two names refer to the same hotel and the presence of errors in the data. In our case, uncertainty is due to the existence of several hotels with similar names in the area we consider. For instance, consider the following hotel names ”Grand Hotel”, ”Soho Grand Hotel” and ”Tribeca Grand Hotel”. Are these the names of three different hotels or of only two different hotels? Another case of uncertainty is when a hotel has more than one name. In the Soho area, the hotel named ”Howard Johnson Express Inn” according to Google Earth, is named ”Metro Three Hotel Llc” in Yahoo Maps, and indeed these are two names of the same hotel.

[4] http://earth.google.com
[5] http://maps.yahoo.com



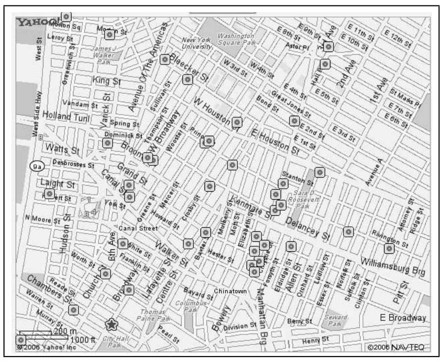


Fig. 1. A Map from Google Earth.

Fig. 2. A map from Yahoo Maps.

In this work we propose the following three-step integration process. *(1)* Retrieve the maps, extract relevant objects from the maps and compute the location of the objects. *(2)* Apply a matching algorithm for finding pairs of corresponding objects. *(3)* Display objects to the user (or return them as a dataset) where each pair of corresponding objects is represented by a single object. Objects that do not belong to any pair of corresponding objects may also be presented.

We now illustrate these steps using the Soho-hotels scenario. Initially, a search for hotels in Soho, New-York was made in both Google Earth and Yahoo Maps, and two result images were retrieved (the images shown in Fig. 1 and Fig. 2). The two images that were found in the search were oriented using geo-referencing. Then, geographical objects were generated by digitizing the maps, that is, by identifying in the raster images icons of hotels and calculating the locations of the hotels based on the geo-referencing. In this example scenario, hotel names were inserted by a human user. In the future we expect many maps on the Web to be in formats that computers can easily process without the need of human intervention. GML (Geographic Markup Language) [1] is an example of such a format.

The second step was to apply a matching algorithm to the two datasets that were extracted from the maps. The result of this step consists of pairs of objects that represent the same hotel, and of singletons representing hotels that appear in only one of the sources. More details about the matching algorithm will be given in the next section.

The final step of the integration is displaying to the user the pairs and singletons produced by the matching algorithm. Before providing the results, conditions can be used for selecting which objects to display. Note that filtering the results at this step makes it possible to apply conditions that use attributes from both sources.

3 Matching Algorithms

The most involved part of an integration process is the discovery of corresponding objects, *i.e.,* the matching algorithm. Several matching algorithms that use only the location of objects were proposed in the past [2, 3]. We now present three new algorithms that are built upon existing location-based algorithms and use attributes of objects for improving the matching.

3.1 Framework

First, we present our framework. A *dataset* is a collection of geographical objects that are extracted from a given map. Each object represents a single real-world *geographical entity* and has a point location. (For an object that has a polygonal shape, we consider the center of mass of the polygonal shape to be the point location of the object.) The distance between two objects is the Euclidean distance between their point locations. We denote by *distance*(a, b) the distance between two objects a and b.

An object may have, in addition to location, attributes that contain information about the entity that the object represents. We distinguish between two types of attributes. An attribute I of objects in a dataset A is *unique* if every two objects in A have different values for I, *i.e.*, I is a candidate key. We consider I as *non-unique* if there can be two objects in A that have the same value for I. For example, in a dataset of hotels, the name of a hotel is a unique attribute, since it is unlikely that two hotels in the same vicinity will have the same name. We consider rating (number of stars) as non-unique, because two proximate hotels may have the same number of stars. When locations of objects are not accurate, we can improve a basic matching algorithm by using additional attributes. If the additional information is correct, a unique attribute can be used for discovering pairs of corresponding objects that the basic algorithm fails to match. Both unique and non-unique attributes can be used for detecting pairs of non-corresponding objects that are, wrongly, deemed corresponding by a matching algorithm.

In integration of maps, locations of objects are not accurate, because the process of extracting objects and computing their locations, by digitizing an image, introduces errors. Furthermore, maps on the Web may not be accurate to begin with. Thus, given two datasets A and B that are extracted from two maps, two corresponding objects $a \in A$ and $b \in B$ may not have the same location. Yet, for each dataset, errors are normally distributed with some standard deviation σ. So, for 98.8% of the objects, their distance from the real-world entity that they represent is less than or equal to 2.5σ. Hence, for 98.8% of the pairs $\{a, b\}$ of corresponding objects, it holds that *distance*$(a, b) \leq \beta$, where $\beta = \sqrt{(2.5\sigma_A)^2 + (2.5\sigma_B)^2}$ is the *distance bound* of A and B (σ_A and σ_B are the standard deviations of the error distributions in A and B, respectively). In our algorithms, pairs $\{a, b\}$ with *distance*$(a, b) > \beta$ are never deemed corresponding objects.

A matching algorithm receives a pair of datasets A and B and returns two sets P and S. The set P consists of pairs $\{a, b\}$, such that $a \in A$ and $b \in B$ are likely to be corresponding objects. The set S consists of singletons $\{s\}$ (where $s \in A \cup B$) such that, with high likelihood, s does not have a corresponding object. Location-based matching algorithms compute the sets P and S according to the distance between objects.

3.2 The New Matching Algorithms

We now describe three new algorithms that receive an existing matching algorithm $\mathcal{M}$ and improve it by using the information provided by some specified attributes. We divide the input to these algorithm into two parts. One part consists of two datasets A and

```
Pre-D_{[M,X]}(A, B)

Parameters: A matching algorithm $\mathcal{M}$, a set of unique attributes $X$
Input: Datasets $A$ and $B$
Output: A set $P$ of pairs and a set $S$ of singletons
1: $P \leftarrow \emptyset, S \leftarrow \emptyset, A' \leftarrow A, B' \leftarrow B$
2: let $\beta$ be the distance bound of $A$ and $B$
3: for each $a \in A$ and $b \in B$ such that $a.x = b.x$ for some attribute $x \in X$ do
4:     if distance$(a, b) \leq \beta$ then
5:         $P \leftarrow P \cup \{a, b\}$
6:         $A' \leftarrow A' - \{a\}, B' \leftarrow B' - \{b\}$
7: $(P', S') \leftarrow \mathcal{M}(A', B')$
8: $P \leftarrow P \cup P', S \leftarrow S'$
9: return $(P, S)$

Post-R_{[M,X]}(A, B)

Parameters: A matching algorithm $\mathcal{M}$, a set of attributes $X$
Input: Datasets $A$ and $B$
Output: A set $P$ of pairs and a set $S$ of singletons
1: $(P, S) \leftarrow \mathcal{M}(A, B)$
2: for each $\{a, b\} \in P$ such that $a.x \neq b.x$ for some attribute $x \in X$ do
3:     $P \leftarrow P - \{a, b\}$
4: return $(P, S)$

Pre-F_{[M,X,φ]}(A, B)

Parameters: A matching algorithm $\mathcal{M}$, a set of non-unique attributes $X$, a factor $\phi$
Input: Datasets $A$ and $B$
Output: A set $P$ of pairs and a set $S$ of singletons
1: $P \leftarrow \emptyset, S \leftarrow \emptyset$
2: let distance$_n(x, y)$ be a new distance function that, initially, is equal to
   distance$(x, y)$
3: for each $a \in A$ and $b \in B$ such that $a.x \neq b.x$ for some attribute $x \in X$ do
4:     distance$_n(a, b) \leftarrow \phi \cdot$ distance$(a, b)$
5: let $\mathcal{M}_n$ be the matching algorithm $\mathcal{M}$ when run using the distance function
   distance$_n(x, y)$ instead of using the Euclidean distance function distance$(x, y)$
6: $(P, S) \leftarrow \mathcal{M}_n(A, B)$
7: return $(P, S)$
```

Fig. 3. The algorithms Pre-process detection, Post-process removal and Pre-process factorizing

B that should be joined. The second part consists of $\mathcal{M}$, a set X of the given attributes and, for the third algorithm, an additional factor ϕ. We denote by P and S the set of pairs and the set of singletons, respectively, that the algorithms return. The pseudocode of all three algorithms is presented in Fig. 3.

Pre-process detection (*Pre-D*)
The *Pre-D* algorithm uses unique attributes for detecting corresponding objects, and then it calls another matching algorithm on the remaining objects. The algorithm has two steps.

1. For each pair of objects $a \in A$ and $b \in B$, such that a and b have the same value for some unique attribute of X and the distance between them does not exceed the distance bound of A and B, the pair $\{a, b\}$ is added to P, a is removed from A and b is removed from B.
2. The matching algorithm $\mathcal{M}$ is applied to the remaining objects of A and B. Upon termination, the pairs of the result are added to P and the singletons—to S.

Post-process removal (*Post-R*)
The *Post-R* algorithm uses a set of attributes X for detecting pairs of objects that are erroneously matched by another algorithm. The *Post-R* algorithm has two steps.

1. The matching algorithm $\mathcal{M}$ is applied to A and B. The result is a set P of pairs and a set S of singletons.
2. For each pair of objects $\{a, b\}$ in P, such that a and b have different values for some attribute of X, the pair $\{a, b\}$ is removed from P.

Pre-process distance factorization (*Pre-F*)
The *Pre-F* algorithm uses a set X of non-unique attributes as follows. For every pair of objects $a \in A$ and $b \in B$ that have different values for some attribute of X, the distance between a and b is multiplied by the given factor $\phi > 1$. Note that increasing the distance between objects lowers the probability that they will be matched by a location-based algorithm. The algorithm $\mathcal{M}$ uses the new distances to join A and B.

In our experiments, we tested eight different combinations of the above algorithms. Suppose that the set Y contains the shared attributes of two datasets A and B. Let *unique*(Y) and *non-unique*(Y) be the sets of unique and non-unique attributes of Y, respectively. Given a location-based matching algorithm $\mathcal{M}$, the following are the eight possible ways of computing the matching of A and B.

1. Use only the location based algorithm $\mathcal{M}$, *i.e.*, return $\mathcal{M}(A, B)$.
2. Use *Post-R* with $\mathcal{M}$. That is, return $\textit{Post-R}_{[\mathcal{M},Y]}(A, B)$.
3. Use *Pre-D* with $\mathcal{M}$. That is, return $\textit{Pre-D}_{[\mathcal{M},\textit{unique}(Y)]}(A, B)$.
4. Combine *Pre-D* and *Post-R*, *i.e.*, return $\textit{Post-R}_{[\textit{Pre-D}_{[\mathcal{M},\textit{unique}(Y)]},Y]}(A, B)$.
5. Use *Pre-F* with $\mathcal{M}$. That is, return $\textit{Pre-F}_{[\mathcal{M},\textit{non-unique}(Y),\phi]}(A, B)$.
6. Combine *Post-R* with *Pre-F*, *i.e.*, return $\textit{Post-R}_{[\textit{Pre-F}_{[\mathcal{M},\textit{non-unique}(Y),\phi]},Y]}(A, B)$.
7. Combine *Pre-D* with *Pre-F*. That is, return the result of the following expression: $\textit{Pre-D}_{[\textit{Pre-F}_{[\mathcal{M},\textit{non-unique}(Y),\phi]},\textit{unique}(Y)]}(A, B)$.
8. Combine all the three methods by applying *Pre-F*, *Pre-D*, $\mathcal{M}$ and, finally, *Post-R*, *i.e.*, return $\textit{Post-R}_{[\textit{Pre-D}_{[\textit{Pre-F}_{[\mathcal{M},\textit{non-unique}(Y),\phi]},\textit{unique}(Y)]},Y]}(A, B)$.

3.3 Computing the Distance Bound

Applying a matching algorithm requires knowing the distance bound β (or an approximation of it). The approximation of β is computed based on approximations of σ_A and σ_B—the standard deviations of the error distributions in the integrated datasets (see Section 3.1). The values σ_A and σ_B (we also call them the *errors* of the datasets) are sometimes provided with the maps, and in other cases we need to estimate them.

The error of a dataset is caused by errors in the procedure of collecting and processing the geographical data. The procedure is different when generating raster (imagery) maps and when vector (feature based) maps are produced. (See [11] for more detailed descriptions of these procedures.)

Raster maps are typically generated from satellite or aerial photographs. There are three main causes of error in the process of creating raster maps. First, errors are introduced when the photos are orthorectified *i.e.,* when correcting the photos to accurately represent the surface of the earth. Second, the size of the pixels in the photo affects the error. Currently, a resolution of 70cm per pixel at nadir is common in satellite photos (*e.g.,* in the two main high-resolution commercial earth-observation satellites IKonos and QuickBird). The first two factors are relatively small and the main cause of error is the third factor which is the accuracy of the geo-referencing process *i.e.,* the accuracy of matching earth coordinated to the photo. The accuracy of the geo-referencing depends on the existence and accuracy of reference points. When no reference points exist, the accuracy is about 10 meters, while when there are reference points, the accuracy is about 1–10 meters, depends on the accuracy of the reference points. Extracting features from the raster image (*e.g.,* identifying the location of an hotel) also introduces an error which is approximately the number of pixels of the error in the extraction process multiplied by the resolution.

Vector maps are usually created either by governmental mapping agencies, or by commercial companies, according to an agreed mapping standard. The standard defines accuracy requirements that depend on the map scale. Typically, for urban areas, map scales are between 1/1000–1/10000. Normally, the required accuracy for such scales is about 0.3–0.4mm. This means that at a scale of 1/1000, the error is about 0.3–0.4 meters. At a scale of 1/10000, the error is approximately 3–4 meters.

3.4 Measuring the Quality of the Result

We use *recall* and *precision* to measure the accuracy of a matching algorithm. Remember that the result of a matching algorithm consists of sets (singletons and pairs). A set is correct if it is either a pair of corresponding objects or a single object that has no corresponding object. Given the result of a matching algorithm, the recall is the ratio of the number of correct sets in the result to the number of all correct sets. For example, a recall of 0.8 means that 80% of the correct sets appear in the result. The precision is the ratio of the number of correct sets in the result to the number of sets in the result. For example, a precision of 0.9 means that 90% of the sets in the result are correct.

In our experiments, we knew exactly which sets were correct and, hence, were able to determine the precision and recall. For synthetic data, all the information about the data was available to us. For real-world data, we determined the correct sets manually, using all the available information.

4 Experiments

In this section, we describe the results of extensive experiments on both real-world and synthetically generated data. The goal of our experiments was to compare the eight combinations, presented in Section 3.2, over data with varying levels of inaccuracy and incompleteness. We also wanted to determine by how much our methods improve existing location-based algorithms. For that, we tested the effect of our methods on the following three location-based algorithms: nearest-neighbor (NN), mutually-nearest (MUTU) and normalized-weights (NW); see [3] for a description of these algorithms.

4.1 Tests on Real-World Data

We present the results of integrating the maps of hotels in Soho as described in Section 2. The Google-Earth map presents 28 hotels and the map from Yahoo Maps presents 39 hotels and inns. A total number of 44 hotels and inns appear in these sources, where 21 hotels appear in both of the sources while 23 appear in only one source. For both sources, we used an error (σ) of 100 meter because identifying the location of an hotel based on an icon is highly inaccurate.

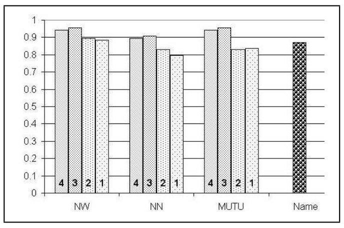


Fig. 4. Tests on real-world data

Figure 4 shows the harmonic mean of the recall and precision (HRP) for the three location-based algorithms (NW, MUTU, NN). Each one of the three algorithms was tested according to the first four combinations of Section 3.2. (The other four combinations are not applicable, since the only attribute, hotel name, is unique.) The third combination, *Pre-D*, is clearly the best for each of the three algorithms. It is slightly better than the fourth combination, which includes both *Pre-D* and *Post-R*, since the attribute hotel name is not always accurate (e.g., one hotel has different names in the two sources). For comparison, Figure 4 also shows the result of matching just according to hotel names. Note that for combinations 2–4, the process was semi-automatic, since hotel names do not appear in Yahoo Maps.

4.2 Tests on Synthetic Data

In order to test our methods on data with varying levels of accuracy and incompleteness, we randomly generated synthetic datasets using a two-step process. First, the real-world entities are generated. The locations of these entities are randomly chosen, according to a uniform distribution, in a square area. Each entity has one unique attribute U and one non-unique attribute N with randomly-chosen values. The non-unique attribute has five possible values (as for the number of stars of a hotel). In the second step, the objects in each dataset are generated. Each object is associated with a distinct entity and its location is chosen with an error that is normally distributed (relative to the location of the entity). In each dataset, different objects correspond to distinct entities. For each object, the attribute U has either the same value as in the corresponding entity, null (for

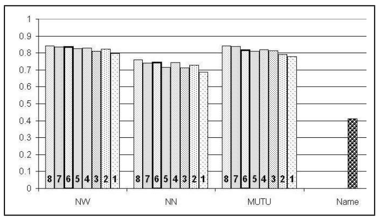


Fig. 5. Results of Test I

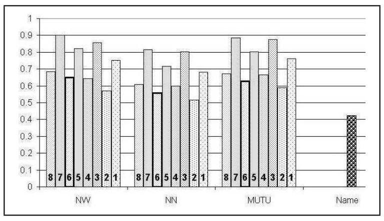


Fig. 6. Results of Test II

incompleteness) or an arbitrary random value (for inaccuracy). We denote by $c(U)$ the percentage of objects that have a non-null value for U and by $a(U)$ the percentage of objects that have either the correct value or null. Values are similarly assigned to N.

We present the results of two tests. In Test I, the values of the attributes are either accurate or missing (i.e., null). In Test II, all the objects have values for U and N, but some of those values are inaccurate. In both tests, there are 1000 entities in a square area of 1350×1350 meters with a minimal distance of 15 meters between entities. Each dataset has 750 objects that are randomly chosen for 750 entities using a standard deviation of $\sigma = 12$ meters for the error distribution. In Test I, the attributes in each dataset have either the correct values or nulls as follows: $a(U) = a(N) = 100\%$, $c(U) = 40\%$ and $c(N) = 60\%$. That is, only 40% of the objects have the correct value for the unique attribute and only 60% of the objects have the correct value for the non-unique attribute (if the value is not the correct one, then it is null). In Test II, attributes always have non-null values but not necessarily the correct ones, i.e., $c(U) = c(N) = 100\%$ and $a(U) = a(N) = 80\%$.

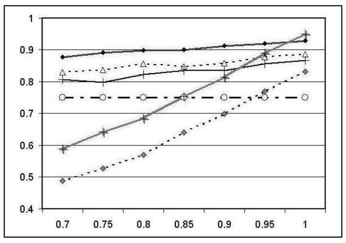
In Test I and Test II, we tried the eight combinations of Section 3.2 with each of the three algorithms. The results, depicted in Fig. 5 and. 6, show the harmonic mean of the recall and precision for the eight combinations involving each algorithm. Each bar is for the combination identified by the number on that bar. For comparison, we also show the result obtained by a matching algorithm that only uses the unique attribute (Name).

Test I shows that when information is partial but accurate, the eighth combination that uses all of the three algorithms (*Pre-D*, *Post-R* and *Pre-F*) is the best. Test II shows that when information is inaccurate, *Post-R* is not effective (as was also the case for the real-world data) and it is better to use just *Pre-D* and *Pre-F* (the seventh combination).

Figures 7 and 8 show the performance of the NW method for varying levels of completeness and accuracy. In Figure 7, the accuracy varies, *i.e.*, $a(U) = a(N) = 70\% \ldots 100\%$, and the completeness is fixed, *i.e.*, $c(U) = c(N) = 100\%$. In Figure 8, the completeness varies, *i.e.*, $c(U) = c(N) = 40\% \ldots 100\%$, and the accuracy is fixed, *i.e.*, $a(U) = a(N) = 100\%$. In each graph, the serial number refers to the combination that produced the graph. Note that the results of only 6 methods (1,2,3,5,7,8) are presented, since the other two are inferior.

The followings are our conclusions from the tests.

1. When there is a *unique* attribute, it is always good to identify pairs and remove them from the matching algorithm (Method 2).



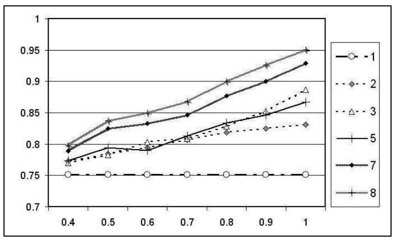


Fig. 7. Results of NW for varying accuracy

Fig. 8. Results of NW for varying completeness

2. When there is a non-unique attribute, it is always good to use factorized distance (Method 5).
3. Although additional information improves the quality of the results, the main factor that determines the quality is still the location-based algorithm.
4. When the attributes are not accurate, using the additional information before the matching improves the quality of the result. But using it after the location-based matching has a negative effect, for the following reason. While there is only a low probability that two proximate yet non-corresponding objects have the same value for a unique attribute,there is a considerably higher probability that two corresponding objects have different values for some unique attribute.

The tests show that in all cases using additional attribute before applying a location-based matching algorithm improves the quality of the results. Applying additional information at the end yields an improvement only if that information is accurate.

5 Conclusion

Traditionally, integration of geo-spatial data is being done using map conflation [13, 6]. However, map conflation is not efficient since whole maps are integrated, not just selected objects. Thus, conflation is not suitable for Web applications or in the context of mediators [4, 12, 19, 20] where users request answers to specific queries. Integrating spatial datasets using only geometrical or topological properties [2, 3, 14] or using only alpha numeric attributes [9, 10], both do not use all the available information but can be combined using the approach we introduced in this paper.

Other approaches use both spatial and non-spatial attributes (e.g. [7, 15, 17]). However, these approaches some time remain on the schema level, rather than actually matching the objects, such as [7], or has large computation time as [15, 17].

In this work we showed how data from maps on the Web can be integrated using location-based algorithms, and how to utilize information additional to location when such information exists. We presented three new matching algorithms and tested them on data with varying levels of incompleteness and inaccuracy. Interestingly, our experiments show that when the additional information is accurate it should be used both before and after the location-based matching process. When the additional information is not very accurate, the information should be used only prior to the location-based matching process. Our experiments show that the new algorithms improve the existing location-based matching algorithms.

References

1. Geographic Markup Language (GML). http://www.opengeospatial.org/standards/gml.
2. 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.
3. C. Beeri, Y. Kanza, E. Safra, and Y. Sagiv. Object fusion in geographic information systems. In *VLDB*, pages 816–827, 2004.
4. O. Boucelma, M. Essid, and Z. Lacroix. A WFS-based mediation system for GIS interoperability. In *ACM-GIS*, pages 23–28, 2002.
5. T. Bruns and M. Egenhofer. Similarity of spatial scenes. In *SDH*, pages 31–42, Delft (Netherlands), 1996.
6. 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.
7. T. Devogele, C. Parent, and S. Spaccapietra. On spatial database integration. In *IJGIS, Special Issue on System Integration*, 1998.
8. F. T. Fonseca and M. J. Egenhofer. Ontology-driven geographic information systems. In *ACM-GIS*, pages 14–19, Kansas City (Missouri, US), 1999.
9. L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.
10. L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an RDBMS for web data integration. In *Proceedings of the 12th international conference on World Wide Web*, pages 90–101, 2003.
11. J. C. McGlone. *Manual of Photogrammetry, Fifth Edition*. American Society of Photogrammetry and Remote Sensing, 2004.
12. Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *VLDB*, pages 413–424, 1996.
13. A. Saalfeld. Conflation-automated map compilation. *IJGIS*, 2(3):217–228, 1988.
14. A. Samal, S. Seth, and K. Cueto. A feature based approach to conflation of geospatial sources. *IJGIS*, 18(00):1–31, 2004.
15. 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.
16. H. Uitermark, P. Van Oosterom, N. Mars, and M. Molenaar. Ontology-based geographic data set integration. In *Proceedings of Workshop on Spatio-Temporal Database Management*, pages 60–79, Edinburgh (Scotland), 1999.
17. V. Walter and D. Fritsch. Matching spatial data sets: a statistical approach. *IJGIS*, 13(5):445–473, 1999.
18. J. M. Ware and C. B. Jones. Matching and aligning features in overlayed coverages. In *ACM-GIS*, pages 28–33, 1998.
19. G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.
20. G. Wiederhold. Mediation to deal with heterogeneous data sources. In *Introperating Geographic Information Systems*, pages 1–16, 1999.