Finding Corresponding Objects when Integrating Several Geo-Spatial Datasets

Catriel Beeri
The Hebrew University
Jerusalem, Israel
cbeeri@cs.huji.ac.il

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

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

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

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

ABSTRACT

When integrating geo-spatial datasets, a join algorithm is used for finding sets of corresponding objects (i.e., objects that represent the same real-world entity). Algorithms for joining two datasets were studied in the past. This paper investigates integration of three datasets and proposes methods that can be easily generalized to any number of datasets. Two approaches that use only locations of objects are presented and compared. In one approach, a join algorithm for two datasets is applied sequentially. In the second approach, all the integrated datasets are processed simultaneously. For the two approaches, join algorithms are given and their performances, in terms of recall and precision, are compared. The algorithms are designed to perform well even when locations are imprecise and each dataset represents only some of the real-world entities. Results of extensive experiments show that one of the algorithms has the best (or close to the best) performances under all circumstances. This algorithm has a much better performance than applying sequentially the one-sided nearest-neighbor join.

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

General Terms:
Algorithms, Experimentation

Keywords:
Location-based join, geospatial datasets, spatial join, corresponding objects, integration

*This author was supported by an NSERC grant.
†The work of this author was supported by the Israel Science Foundation (Grant No. 96/01).

1. INTRODUCTION

Integration of geo-spatial data from heterogeneous sources has many important applications. One example is combining up-to-date data, say from a satellite image, with data from a map that contains verbal descriptions of entities. When geographical entities are represented in different sources, and each source stores different properties of the entities, integration makes it possible to obtain all the available information on each entity.

Integration is essentially a join of datasets. The main task in a join is to find all sets of *corresponding objects*, i.e., objects that represent the same real-world entity in distinct sources. Over heterogeneous sources, however, finding corresponding objects is difficult, since there are no global identifiers. In principle, both spatial and non-spatial properties may be used, in lieu of global identifiers, for integrating geographical data. However, only location is always available for spatial objects. Thus, we investigate *location-based join*.

Since in many cases, locations uniquely identify objects in a dataset, location-based join seems to be an easy task. This is not so, however, for several reasons. First, measurements introduce errors, and the errors in different datasets are independent of each other. Second, each organization has its own approach and requirements. Hence, different organizations use different measurement techniques and may record spatial properties of entities using a different scale or a different structure. For example, one organization might represent buildings as points, while another could represent them as polygons. While an estimated point location can be derived from a polygonal shape, it may not agree with a point location in another database. A third reason could be displacements caused by cartographic generalizations.

For the above reasons, location-based joins do not provide a precise answer, but rather an approximation. The quality of the approximation is determined by characteristics of the joined datasets, such as sizes of the errors, the density of objects, and the relative overlap.

In this paper, we introduce algorithms for location-based join of three or more sources, under the assumptions that locations are given as points and each dataset has at most one object per real-world entity. The rationale underlying all our algorithms is that even in the presence of measurement errors, corresponding objects should have close locations.

For location-based join of two datasets, the current state of the art is the *one-sided nearest-neighbor join (nn-join)* [13] that joins an object from one dataset with its closest neighbor in the other dataset (see also Section 3.2 below). Beeri et al. [1] showed that the nn-join has several disadvantages, and they introduced three algorithms that outperform the nn-join, namely, the *mutually-nearest join*, the *probabilistic join* and the *normalized-weights join*.

In this paper, we consider the problem of computing sets of corresponding objects when more than two sources are at hand. Two basic approaches are presented and compared: the *sequential approach* and the *holistic approach*. In the sequential approach, a join algorithm for two datasets is applied sequentially. In the holistic approach, the join is applied to all the sources simultaneously. For the holistic approach, we had to develop new algorithms. The new algorithms are designed for a join of three datasets, but can be easily extended to a join of any number of datasets.

It may seem that the sequential approach is problematic, since sources that appear earlier in the sequence may have greater influence on the result than sources that appear towards the end of the sequence. Furthermore, an error in one of the joins of the sequence may cause additional errors in subsequent joins. Our experiments, however, show that, in many cases, there is a sequence of joins that produces a good approximation of the sets of corresponding objects.

We tested our algorithms on both real-world data that describe hotels in Tel-Aviv, and randomly generated data. We conducted our tests in varying conditions of the measurement errors, the density of each geo-spatial dataset and the degree of overlap between the tested datasets. The results of our tests illustrate the strengths and weaknesses of the holistic and sequential approaches.

The main contribution of our work is in showing that point locations can be effectively used for finding corresponding objects, even when more than two sources are present. Since locations are always available for spatial objects, location-based joins provide a practical approach to the integration of geographic datasets. Additional properties of objects that may be available (e.g., names or polygonal locations) can be used to enhance the strength of our algorithms.

2. TERMINOLOGY AND ASSUMPTIONS

In this section, we present the notion of a join algorithm and describe the results of such an algorithm. In addition, we discuss the quality of the result and the factors that influence this quality.

2.1 Join Algorithms and Their Results

A *geo-spatial dataset* stores *spatial objects* and each spatial object represents a single real-world *geographic entity*. To abbreviate the terminology, we use the terms *dataset*, *object* and *entity*. In each dataset there is at most one object for each real-world entity. An object has associated spatial and non-spatial attributes. Spatial attributes describe the location, height, shape and topology of an entity. Examples of non-spatial attributes are building name, accessibility to people in a wheelchair, number of rooms in a hotel, etc.

When geographic databases are integrated, the main task is to identify sets of objects that represent the same real-world entity in different sources. A *join algorithm* receives datasets as input and it generates *join sets*. Each join set contains at most one object from each dataset. A join set J is *correct* if it consists exactly of all the objects that correspond to some entity e; that is, there exists an entity e, such that all the objects that represent e are in J and there are no objects in J that do not represent e.

In this paper, we investigate join algorithms under the following assumptions. First, we assume that locations of objects are recorded as points. More complex forms of recording locations (e.g, polygons) can be approximated by points (e.g., by computing the center of mass). Second, in each dataset, distinct objects represent distinct real-world entities. This is a realistic assumption for many GIS applications. Finally, we consider only join algorithms that use locations of objects, but no additional properties of those objects. As already mentioned, location-based join is a nontrivial task. Understanding the factors that determine the quality of location-based join algorithms is a basis for developing join algorithms that use all available properties.

2.2 Quality of Results

2.2.1 *Measuring the Quality*

As in information retrieval, we measure the quality of a join algorithm in terms of *recall* and *precision*. Note that applying these measures requires full knowledge of the correspondence between objects and the entities they represent. In practical circumstances, this knowledge is not available; however, it was available for the datasets that we used in our tests. Hence, we could use recall and precision to describe the quality of our test results.

The straightforward definition of recall and precision is based on the number of correct sets. Recall is the percentage of correct join sets that actually appear in the result (e.g., 86% of all the correct join sets appear in the result). Precision is the percentage of correct join sets out of all the join sets in the result (e.g., 91% of the sets in the result are correct). The problem with this definition is that it does not distinguish between two sets, such that one consists mostly of objects that correspond to the same entity while in the other set, each object corresponds to a different entity. Intuitively, we would like to consider the first set as having a higher level of correctness.

Therefore, we use a definition that counts the number of correct pairs rather than the number of correct sets. In other words, we count pairs of corresponding objects. Since entities are not necessarily represented in all the sources, this counting requires some care.

Consider n datasets $D_1, \ldots, D_n$ and let $\perp_1, \ldots, \perp_n$ be n fresh symbols. For each $1 \leq i \leq n$, we call $\perp_i$ the *null value* of D_i. A *pair* is either a pair of objects from two distinct D_i's or an object o_i from some D_i and a null value $\perp_j$, where $j \neq i$. In the first case, the pair is *correct* if it consists of two corresponding objects. In the second case, it is correct if o_i has no corresponding object in D_j. Note that pairs are unordered, that is, (a, b) is the same as (b, a) and hence is counted just once.

Consider a join set J over $D_1, \ldots, D_n$. For each $1 \leq i \leq n$, we add to J the null value $\perp_i$ if J does not include any object of D_i. The result is the set $\bar{J}$. We denote by *pairs*$(\bar{J})$ the set of all pairs that are obtained from $\bar{J}$.

Let $J_1, \ldots, J_m$ be the join sets that are produced by some join algorithm for the datasets $D_1, \ldots, D_n$. The set $\bar{\mathcal{J}}$ consists of all distinct pairs obtained from $\bar{J}_1, \ldots, \bar{J}_m$ i.e., $\bar{\mathcal{J}} = \cup_{i=1}^{m} \mathit{pairs}(\bar{J}_i)$. Suppose that there are k real-world entities $e_1, \ldots, e_k$. Let $\bar{E}_1, \ldots \bar{E}_k$ be the perfect result for the datasets $D_1, \ldots, D_n$; that is, $\bar{E}_i$ comprises all the objects corresponding to e_i and the null values from the datasets D_j that do not contain any object for e_i. The set $\bar{\mathcal{E}}$ consists of all distinct pairs obtained from the perfect result, i.e., $\bar{\mathcal{E}} = \cup_{i=1}^{k} pairs(\bar{E}_i)$; note that all these pairs are correct.

Let P_r be the number of pairs in $\bar{\mathcal{J}}$, let P_c be the number of correct pairs in $\bar{\mathcal{J}}$ and let P_e be the number of pairs in $\bar{\mathcal{E}}$. For the result $J_1, \ldots, J_m$, the precision is P_c/P_r and the recall is P_c/P_e.

In all our experiments, including those that are presented in this paper and those that are not, the two definitions of recall and precision ranked join algorithms similarly. In this paper, we only present results using the pair-based measure since, as explained earlier, it is more sensitive to small changes in the input.

2.2.2 *Factors Affecting Recall and Precision*

Several factors may influence the quality of results of a join algorithm. One of these is the distribution of errors in each dataset. Section 3.1 describes how this distribution is used in join algorithms.

The *density* of a dataset is the number of objects per unit of area. Each dataset has an *error* which is the maximal distance between an entity and its representing object. The *choice factor* is the number of objects in a circle with a radius that is equal to the error (note that the choice factor is the product of the density and the area of that circle). Intuitively, for a given entity, the choice factor is an estimate of the number of objects in a dataset that could possibly represent that entity. When the choice factor is large, it is difficult to achieve high-quality results. Note that the above factors need not be uniform in the geographic area that is represented by a given dataset.

Another important factor is the *overlap* between datasets. If two datasets A and B have m and n objects, respectively, and there are precisely c entities that are represented in both sets, then the overlap between A and B is defined to be $\sqrt{\frac{c}{m} \cdot \frac{c}{n}}$. The overlap is a measure of the fraction of objects that have a corresponding object in the other set. One of the challenges we faced was to develop an algorithm that has high recall and precision for varying degrees of overlap between datasets.

3. THE SEQUENTIAL APPROACH

In this section, we describe the *sequential approach* for joining three datasets. In this approach, we use a 2-join algorithm, i.e., a join algorithm for two datasets, and we apply it twice—first we join two of the given datasets and then we join the result of the first join with the third dataset.

Applying the sequential approach necessitates answering the following questions. First, which 2-join algorithm should we use? Second, how does the error distribution affect different 2-join algorithms? Third, assume that two datasets have already been joined. The result of this join is viewed as a (virtual) dataset and it should be joined with the third dataset. For that, we need to decide what is the location of objects and what is the estimated error in the result of the first join. In this section, we answer these questions.

3.1 Accuracy Parameters

First, we briefly consider accuracy parameters for a single dataset. In a given dataset, the exact error in the locations of objects is normally distributed with a standard deviation σ and a mean equal to zero. We assume that the error of the dataset has a fixed size and we take it to be $m = 2.5\sigma$.

When $m = 2.5\sigma$, then for 98.8% of the objects, the distance between an object and the entity that the object represents is smaller than m. Generally, this percentage increases when m is increased, and decreases when m is decreased.

When datasets A and B are joined by a 2-join algorithm, their *mutual-error bound*, denoted by β_{AB}, is

$$\beta_{AB} = \sqrt{{m_A}^2 + {m_B}^2}.$$

The mutual-error bound determines the maximal expected distance between a pair of corresponding objects from A and B. Its meaning is analogous to the errors m_A and m_B. That is, for approximately 98.8% of all pairs of corresponding objects, the distance between these objects is smaller than β_{AB}. When a 2-join algorithm is applied to datasets A and B, we assume that a pair of objects, $a \in A$ and $b \in B$, cannot correspond to the same entity if the distance between them is greater than β_{AB}. Consequently, the distance between these two objects is taken to be ∞.

Now, let $\mathcal{J}(A, B)$ be the result of joining datasets A and B. We describe how this result can be viewed as a dataset. For that, we need to determine the objects in this result, the location of each object, and the error m_{AB} for the result. In $\mathcal{J}(A, B)$, there is an object for each join set that is generated from A and B. If a join set that is produced from A and B is a *singleton* $\{o\}$ (i.e, contains a single object), then the dataset $\mathcal{J}(A, B)$ contains the object o, with its original location. If a join set is a pair $\{a, b\}$, where $a \in A$ and $b \in B$, then $\mathcal{J}(A, B)$ contains a (virtual) object o_{ab}.

As for the location of o_{ab}, a simple approach is to take the average of the locations of a and b. It is customary, however, to take a weighted average, where the weight of a dataset is its *accuracy* [12]. For a dataset A, its accuracy, denoted by w_A, is $\left(\frac{1}{\sigma_A}\right)^2$. Intuitively, we consider a dataset as accurate if most errors are small. That is, the smaller the variance, the higher the accuracy.

Suppose that the (vector) locations of a and b are l_a and l_b, respectively. The location of o_{ab} is the vector l_{ab}, where

$$l_{ab} = \frac{l_a \cdot w_A + l_b \cdot w_B}{w_A + w_B}. \tag{1}$$

The location l_{ab} is on the line that connects the locations of a and b. If the dataset A is more accurate than B, then the new location is closer to l_a, and vice versa. Intuitively, this is so because the location of an object (with respect to the entity that it represents) is likely to be more accurate if it belongs to the more accurate dataset.

Next, consider the error. The error m_{AB} of the dataset $\mathcal{J}(A, B)$ is given by the following formula.

$$\left(\frac{1}{m_{AB}}\right)^2 = \left(\frac{1}{m_A}\right)^2 + \left(\frac{1}{m_B}\right)^2$$

The above formula is equivalent to the next one.

$$m_{AB} = \sqrt{\frac{{m_A}^2 \cdot {m_B}^2}{{m_A}^2 + {m_B}^2}} \tag{2}$$

In the rest of this section, we briefly review three 2-join algorithms that are used in the sequential approach. For more details on these algorithms, see [1].

3.2 The One-Sided Nearest-Neighbor Join

The *one-sided nearest-neighbor join* is commonly used in commercial geographic-information systems [13]. Given an object $a \in A$, we say that an object $b \in B$ is the *nearest B-neighbor* of a if b is the closest object to a among all the objects in B. The one-sided nearest-neighbor join is asymmetric. So, starting with a dataset A and joining B to A, the result consists of all join sets $\{a, b\}$, such that $a \in A$, $b \in B$ and b is the nearest B-neighbor of a.

By the above definition, the result has three properties. First, all the join sets have two objects. Second, every $a \in A$ is in one of the join sets. Third, an object of B may appear in zero, one or more join sets. In order to boost up the recall and precision of this method, and consistently with our approach of how the mutual-error bound is used, we modify the above definition as follows. First, we remove from the result pairs $\{a, b\}$ with $\mathit{distance}(a, b) > \beta_{AB}$, where $\mathit{distance}(a, b)$ denotes the distance between the locations of objects a and b. Then, we add to the result the singleton sets $\{a\}$ and $\{b\}$, for every $a \in A$ and $b \in B$ that do not appear in any pair.

As noted above, the nearest-neighbor join is asymmetric: the results of joining B to A is often different from the result of joining A to B. In the sequential approach, we employ the simple heuristic of always starting with the larger dataset and joining the smaller one, since it produces better results than joining the larger dataset to the smaller one. If we start with the smaller dataset, each object has near it many objects of the larger set, and thus, the choice factor is large. Starting with the larger dataset decreases the choice factor.

3.3 The Mutually-Nearest Join

We say that two objects, $a \in A$ and $b \in B$, are *mutually nearest* with respect to A and B if a is the nearest A-neighbor of b and, in addition, b is the nearest B-neighbor of a. In the *mutually-nearest join,* a two-element join set is created for each pair of mutually-nearest objects that have a distance of β_{AB} or less between them. The intuition behind the mutually-nearest join is that corresponding objects are likely to be mutually nearest. A singleton join set is created for each object that is not in any pair of mutually-nearest objects. Note that it is possible that b is the nearest B-neighbor of a and a is not the nearest A-neighbor of b. In this case, a will be a singleton.

3.4 The Normalized-Weights Method

The normalized-weights method used in the sequential algorithm is an improved version of the method presented in [1]. Consider two datasets $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_n\}$. For each object $a \in A$, we define the function $P_a : B \rightarrow [0, 1]$ that assigns to each $b \in B$ the probability (or likelihood) that *a chooses b* among all the objects of B. Similarly, for each $b \in B$, we define the function $P_b : A \rightarrow [0, 1]$.

Formally, the probability function P_{a_i} is defined, as shown below, in terms of three parameters: the distance, the *distance decay factor* $\alpha > 0$ and the mutual-error bound β_{AB} (the probability function P_{b_j} is defined similarly).

$$P_{a_i}(b_j) = \frac{\mathit{distance}(a_i, b_j)^{-\alpha}}{\sum_{k=1}^{n} \mathit{distance}(a_i, b_k)^{-\alpha} + \beta_{AB}{}^{-\alpha}} \tag{3}$$

There is also the probability that a_i does not choose any object of B, i.e., the probability that a_i will be in a singleton join set of the result. This probability is as follows.

$$P_{a_i}(\perp_B) = \frac{\beta_{AB}{}^{-\alpha}}{\sum_{k=1}^{n} \mathit{distance}(a_i, b_k)^{-\alpha} + \beta_{AB}{}^{-\alpha}} \tag{4}$$

Thus, the sum of the probabilities that a_i chooses one of the b_j's or chooses being a singleton is 1.

The mutual-error bound β_{AB} has the following effect in Formula 3. If $\mathit{distance}(a_i, b_j) > \beta_{AB}$, then a_i and b_j are not likely to be corresponding objects. Thus, $\mathit{distance}(a_i, b_j)$ is taken to be infinity and $\mathit{distance}(a_i, b_j)^{-\alpha} = 0$. In this case, $P_{a_i}(b_j) = 0$. Note that the denominator of Formula (3) is always greater than 0, assuming that $\beta_{AB} > 0$.

According to Formula 3 and the fact that $\alpha > 0$, the probability that a_i chooses b_j increases when the distance between a_i and b_j decreases. In particular, a_i chooses its nearest B-neighbor with the highest probability. The parameter α determines the rate of decrease in the probability as the distance increases. In our tests, we used $\alpha = 2$.

Now, we describe the normalized-weights method. The *matching matrix* M is an $(m + 1) \times (n + 1)$ matrix, such that the element in row i and column j, denoted by μ_{ij}, is defined as follows.

- For $1 \leq i \leq m$ and $1 \leq j \leq n$,
$$\mu_{ij} = P_{a_i}(b_j) \cdot P_{b_j}(a_i).$$
- For $1 \leq i \leq m$ and $j = n + 1$,
$$\mu_{ij} = P_{a_i}(\perp_B) \cdot \prod_{k=1}^{n} (1 - P_{b_k}(a_i)).$$
- For $i = m + 1$ and $1 \leq j \leq n$,
$$\mu_{ij} = P_{b_j}(\perp_A) \cdot \prod_{k=1}^{m} (1 - P_{a_k}(b_j)).$$
- For $i = m + 1$ and $j = n + 1$, $\mu_{ij} = 0$.

Note that in the first case of the above definition, μ_{ij} is assigned the probability that a_i and b_j mutually choose each other. In each row i, the element in the last column $(j = n + 1)$ gives the probability that a_i does not choose any $b \in B$ and is not being chosen by any $b \in B$. Similarly, in each column j, the element in the last row $(i = m + 1)$ gives the probability that b_j does not choose any $a \in A$ and is not being chosen by any $a \in A$.

A row (or a column) r is *normalized* to a value $x > 0$ if the sum s of all the elements of r is equal to x. We can always normalize r to x by dividing each element by $\frac{s}{x}$ (since $s > 0$). The *normalization algorithm* is a sequence of iterations over M. In each iteration, the first m rows are normalized to one and the last row is normalized to the number of objects in A that do not have a corresponding object in B. Then, the first n columns are normalized to one and the last column is normalized to the number of objects in B that do not have a corresponding object in A. When the values for normalizing the last row and last column are not known, we use the mutually-nearest join for computing approximations of these values.

Let $M^{(0)}$ denote the matrix M, as defined above, and let $M^{(k)}$ denote the matrix after k iterations. It was shown by Sinkhorn [18, 19] that the normalization algorithm converges and the result does not depend upon the order of normalizing rows and columns in each iteration. We terminate the

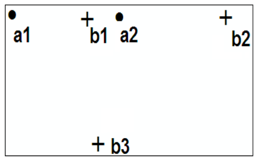


Figure 1: Datasets A (circles) and B (pluses).

normalization algorithm when the sum of each row and each column, except for the last row and the last column, differs from 1 by no more than some very small $\epsilon > 0$.

Let $M^{(t)}$ denote the matrix upon termination of the iterative normalization algorithms. The *confidence* of the join set $\{a_i, b_j\}$ is the value in row i and column j of $M^{(t)}$. The confidence of the join set $\{a_i\}$ is the value in row i and column $n+1$ of $M^{(t)}$. The confidence of the join set $\{b_j\}$ is the value in column j and row $m+1$ of $M^{(t)}$. The result of the normalized-weights method consists of all the join sets with confidence values above a given threshold τ. In general, τ could be given by a user who may wish to control the recall-precision tradeoff. In our tests, we used a threshold that minimized the *error count*. The error count of a join result is the number of objects that do not appear in any join set plus the number of objects that appear in more than one join set. It has been shown in [1] that a threshold that minimized the error count provides a combination of recall and precision that is near the best.

The following example illustrates the normalized-weights method.

EXAMPLE 3.1. Consider two datasets $A = \{a_1, a_2\}$ and $B = \{b_1, b_2, b_3\}$, with a mutual-error bound $\beta_{AB} = 15$. The positions of the objects are shown in Figure 1. The distances between objects are given in the following table.

Distances ($\beta = 15$)

	b_1	b_2	b_3	$\perp_B$
a_1	7	$> \beta$	12.8	15
a_2	3	10	10.2	15
$\perp_A$	15	15	15	

We use the above distances to compute the probability functions, according to Equation 3 and Equation 4. The probability functions P_{a_i} are presented in the next table.

Choice Probabilities for A

	b_1	b_2	b_3	$\perp_B$
P_{a_1}	0.66	0	0.20	0.14
P_{a_2}	0.82	0.07	0.07	0.03

The following table shows the probability functions P_{b_j}.

Choice Probabilities for B

	P_{b_1}	P_{b_2}	P_{b_3}
a_1	0.15	0	0.30
a_2	0.82	0.69	0.48
$\perp_A$	0.03	0.31	0.22

The initial weights in the matching matrix are as follows.

Initial Weights

	b_1	b_2	b_3	$\perp_B$
a_1	0.10	0	0.06	0.09
a_2	0.67	0.05	0.03	0.00
$\perp_A$	0.00	0.28	0.16	0

We now show the first iteration of the normalization algorithm. For each row in the matching matrix, we first compute the sum of all the elements in that row and then divide each element by that sum. For simplicity, in this example, we do not normalize the last row and the last column.

Normalizing the Rows

	b_1	b_2	b_3	$\perp_B$	Sum
a_1	0.41	0	0.24	0.35	1.00
a_2	0.89	0.07	0.04	0.00	1.00
$\perp_A$	0.00	0.28	0.16	0	
Sum	1.29	0.35	0.45		

Now, for each column, we divide every element by the sum of all elements in that column.

Normalizing the Columns

	b_1	b_2	b_3	$\perp_B$	Sum
a_1	0.31	0	0.54	0.35	1.20
a_2	0.68	0.19	0.10	0.00	0.98
$\perp_A$	0.01	0.81	0.36	0	
Sum	1.00	1.00	1.00		

This completes the first iteration. In this example, the normalization algorithm terminates after 9 iterations and returns the following matrix.

After Normalization

	b_1	b_2	b_3	$\perp_B$	Sum
a_1	0.27	0	0.47	0.26	1.00
a_2	0.72	0.18	0.10	0.00	1.00
$\perp_A$	0.01	0.82	0.43	0	
Sum	1.00	1.00	1.00		

For a threshold $\tau = 0.45$, the algorithm returns the join sets $\{a_1, b_3\}$, $\{a_2, b_1\}$ and $\{b_2\}$. If a threshold of $\tau = 0.6$ is used, then only the sets $\{a_2, b_1\}$ and $\{b_2\}$ are returned.

4. THE HOLISTIC APPROACH

In this section, we present two join methods that use the holistic approach. The methods are designed for joining three datasets, but can be easily modified to join any number of datasets. One method is the holistic version of the mutually-nearest method and the other is the holistic version of the normalized-weights method.

4.1 The Holistic Mutually-Nearest Method

The holistic version of the mutually-nearest method is a three-step algorithm. Suppose that A, B and C are the datasets that should be joined. In Step 1, a join set is created for each three objects $a \in A$, $b \in B$ and $c \in C$, such that every two objects in $\{a, b, c\}$ are mutually nearest and the distance between them does not exceed the mutual-error bound (i.e., *distance*$(a, b) \le \beta_{AB}$, etc.) That is, the objects a and b are mutually nearest w.r.t. (with respect to) A and B; the objects a and c are mutually nearest w.r.t. A and C; and b and c are mutually nearest w.r.t. B and C.

In Step 2, we add join sets that contain pairs of mutually nearest objects. We add only some of the mutually-nearest pairs, because an object should not appear in more than one set of the result. Thus, we add a pair of objects $a \in A$ and $b \in B$ to the result when the following four conditions hold. First, the objects a and b are mutually nearest w.r.t. A and B. Second, *distance*$(a, b) \le \beta_{AB}$. Third, no set of triplets that was added in Step 1 contains either a or b. Fourth, there is no object $c \in C$, such that either a and c are mutually nearest and $\mathit{distance}(a,c) < \mathit{distance}(a,b)$ or b and c are mutually nearest and $\mathit{distance}(b,c) < \mathit{distance}(a,b)$. Pairs of objects from A and C or from B and C are added under similar conditions.

In Step 3, a singleton join set is created for each object that does not belong to any set that was created in either Step 1 or Step 2. Note that indeed any object of either A, B or C appears in exactly one join set.

4.2 The Holistic Normalized-Weights Method

There are three variants of the holistic normalized-weights method. In all three variants, first we construct a three-dimensional matrix, such that each possible join set has a corresponding element in the matrix. Then, an iterative normalization is applied to the matrix. The three variants differ in how the normalization is performed; however, in all of them, at the end of the normalization, the elements of the matrix are the confidence values of their corresponding join sets. The method returns the sets with confidence values that exceed a given threshold value.

We now describe the algorithm in more detail. We start by describing the initial matrix. Consider three datasets $A = \{a_1, \ldots, a_l\}$, $B = \{b_1, \ldots, b_m\}$ and $C = \{c_1, \ldots, c_n\}$. The six probability functions $P_{a_i}(b_j)$, $P_{b_j}(a_i)$, $P_{a_i}(c_k)$, $P_{c_k}(a_i)$, $P_{b_j}(c_k)$ and $P_{c_k}(b_j)$ are defined as in Section 3.4. The matching matrix M is an $(l+1) \times (m+1) \times (n+1)$ matrix, such that the element in position (i,j,k), denoted by μ_{ijk}, is defined as follows.

For $1 \le i \le l$, $1 \le j \le m$ and $1 \le k \le n$, the element μ_{ijk} corresponds to the set $\{a_i, b_j, c_k\}$ and

$$\begin{aligned}\mu_{ijk} &= P_{a_i}(b_j) \cdot P_{b_j}(a_i) \cdot \\ &\quad P_{a_i}(c_k) \cdot P_{c_k}(a_i) \cdot \\ &\quad P_{b_j}(c_k) \cdot P_{c_k}(b_j)\end{aligned}$$

That is, μ_{ijk} is the probability that every two elements, among a_i, b_j and c_k, choose each other.

For $1 \le i \le l$, $1 \le j \le m$ and $k = n+1$, the element μ_{ijk} corresponds to the set $\{a_i, b_j\}$ and

$$\begin{aligned}\mu_{ijk} &= P_{a_i}(b_j) \cdot P_{b_j}(a_i) \cdot \\ &\quad P_{a_i}(\bot_C) \cdot \prod_{h=1}^{n}(1 - P_{c_h}(a_i)) \cdot \\ &\quad P_{b_j}(\bot_C) \cdot \prod_{h=1}^{n}(1 - P_{c_h}(b_j))\end{aligned}$$

That is, μ_{ijk} is the probability that a_i and b_j mutually choose each other and, in addition, a_i and b_j do not choose any object of C and are not chosen by any $c \in C$. By symmetry, μ_{ijk} is defined similarly in the following cases.

- $1 \le i \le l$, $j = m+1$ and $1 \le k \le n$ (for $\{a_i, c_k\}$).
- $i = l+1$, $1 \le j \le m$ and $1 \le k \le n$ (for $\{b_j, c_k\}$).

For $1 \le i \le l$, $j = m+1$ and $k = n+1$, the element μ_{ijk} corresponds to the singleton $\{a_i\}$ and

$$\begin{aligned}\mu_{ijk} &= P_{a_i}(\bot_B) \cdot \prod_{g=1}^{m}(1 - P_{b_g}(a_i)) \cdot \\ &\quad P_{a_i}(\bot_C) \cdot \prod_{h=1}^{n}(1 - P_{c_h}(a_i))\end{aligned}$$

That is, μ_{ijk} is the probability that a_i chooses neither an object of B nor an object of C and a_i is not chosen by any $b \in B$ or $c \in C$. The element μ_{ijk} is similarly defined in the following two cases.

- $i = l+1$, $1 \le j \le m$ and $k = n+1$ (for $\{b_j\}$).
- $i = l+1$, $j = m+1$ and $1 \le k \le n$ (for $\{c_k\}$).

Finally, $\mu_{ijk} = 0$, for $i = l+1$, $j = m+1$ and $k = n+1$.

Next, we present the three variants for normalizing the matrix. We use the following simple notation to denote sets of elements of M. The set comprising the element μ_{ijk} is denoted by $M(i,j,k)$. We generalize this notation by writing a range of values instead of just a single value. For example, $M(i, 1\cdots m, 1\cdots n)$ is the set of all elements, such that the first index is equal to i while the second and third indices take all possible values in the ranges $1, \ldots, m$ and $1, \ldots, n$, respectively. In set notation,

$$M(i, 1\cdots m, 1\cdots n) = \{\mu_{ijk} \mid 1 \le j \le m \text{ and } 1 \le k \le n\}.$$

We use $*$ to denote the ranges $1\cdots l+1$, $1\cdots m+1$ and $1\cdots n+1$ for the first, second and third indices, respectively.

Basic Normalization: A *plate* of the matrix M comprises all the elements corresponding to join sets that contain a given object. That is, the plate for $a_i \in A$ is $M(i,*,*)$, the plate for $b_j \in B$ is $M(*,j,*)$, and the plate for $c_k \in C$ is $M(*,*,k)$.

A plate is *normalized* to the value x if the sum s of all its elements is equal to x. Normalizing a plate to x, when $s > 0$, is multiplying each element of it by $\frac{x}{s}$. The *basic normalization* is a sequence of iterations over M, such that in each iteration, each of the $l+m+n$ plates is normalized to 1. The intuition behind normalizing the plates to 1 is that each object is contained in exactly one correct join set.

Complete Normalization: Let $x, y \in \{A, B, C\}$, where $x \ne y$. P_{xy} is the number of correct join sets of size 2 that contain an object from x and an object from y. Similarly, S_x is the correct number of join sets of size 1 (i.e., singletons) that contain an object from x.

In a *complete normalization*, in each iteration all the plates of M are normalized to 1, as in the basic normalization. In addition, in each iteration the following sets are normalized. The set $M(1\cdots l, 1\cdots m, n+1)$ is normalized to the number P_{AB}. The set $M(1\cdots l, m+1, 1\cdots n)$ is normalized to P_{AC}. The set $M(l+1, 1\cdots m, 1\cdots n)$ is normalized to P_{BC}. The set $M(1\cdots l, m+1, n+1)$ is normalized to S_A. The set $M(l+1, 1\cdots m, n+1)$ is normalized to S_B. Finally, the set $M(l+1, m+1, 1\cdots n)$ is normalized to the number S_C.

Note that in real applications, P_{xy} and S_x are not known. The method can still be used in simulations, where the correct values are known. It thus provides a yardstick against which other methods can be compared. As for real applications, the following variant uses heuristics to approximate the unknown values.

Approximate Normalization: In this method, an approximation of P_{xy} and S_x is computed from the result of the holistic normalized-weights method using the basic-normalization variant, as described above. Then, complete normalization, using the approximate values, is applied to the matching matrix.

Let $M^{(0)}$ denote the matrix M, as it was defined above, and $M^{(k)}$ denote the matrix after k iterations. We terminate

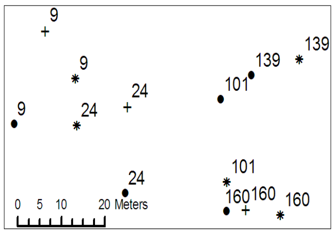


Figure 2: A fragment of a randomly generated test.

the iterative normalization when the sum of elements in each plate is different from 1 by at most some very small $\epsilon > 0$.

Let $M^{(t)}$ denote the matrix upon termination of the iterative normalization. The *confidence* of the join set $\{a_i, b_j, c_k\}$ is the value $\mu_{i,j,k}$ of $M^{(t)}$. The confidence of the join set $\{a_i, b_j\}$ is the value $\mu_{i,j,n+1}$ of $M^{(t)}$. The confidence of the join set $\{a_i\}$ is the value $\mu_{i,m+1,n+1}$ of $M^{(t)}$. Confidence values for other combinations are defined similarly. The result of the normalized-weights method consists of all the join sets with confidence values above a given threshold τ. In general, τ could be given by a user who wants to control the recall-precision tradeoff. In our tests, we used a threshold that minimized the error count, as explained in Section 3.4.

5. TESTING THE METHODS

We tested the different join methods on both randomly generated datasets and real-world datasets. In this section, we present our experiments and use the experimental results to compare the methods.

5.1 Tests on Random Datasets

Our experiments on random datasets are trying to answer the following two questions. First, what is the influence of the density, the choice factor and the overlap on the results of the join algorithms? Second, knowing the density and the overlap of the three datasets, what is the best algorithm for computing their join?

5.1.1 Random-Dataset Generator

There are not sufficiently many real-world datasets to test our algorithms under varying degrees of density and overlap. Moreover, in real-world datasets, it is not always possible to determine accurately the correspondence between objects and real-world entities. Thus, we implemented a random-dataset generator. Our generator is a two-step process. First, the real-world entities are generated. The locations of the entities are randomly chosen and are uniformly distributed in a square area. In the second step, the objects in each dataset are generated. Each object is associated with a distinct entity. The location of an object is defined by a random vector whose origin is the point location of the entity. The size of the vector (i.e., the distance between the object and the entity) is randomly chosen according to a normal distribution, and the angle of the vector is randomly chosen according to a uniform distribution. Locations in each dataset are chosen independently of the other datasets.

The user provides the following parameters to the dataset generator. For the entities—their number, the size of the square area in which they are located and the minimal distance between entities; for each datasets—the number of objects in it and the standard deviation of the error. These parameters allow a user to generate tests with different degrees of density and overlap.

5.1.2 Test Results

We have experimented with many randomly generated datasets, but in this section we only describe a few tests that demonstrate the main conclusions about the performance of each method. In all the tests, entities were generated and their locations were randomly chosen in a square area of 900×900 square meters. For the first six tests, 200 entities were generated, and for the other three tests, we generated 300 entities. The minimal distance between entities was set to 15 meters. In all the datasets, the error is $m = 2.5\sigma$ (i.e., for about 98.8% of the objects, the distance between the object and its corresponding entity is less than m). A fragment of a randomly generated test is presented in Figure 2. This figure illustrates the complexity of computing correct join sets over the randomly generated data.

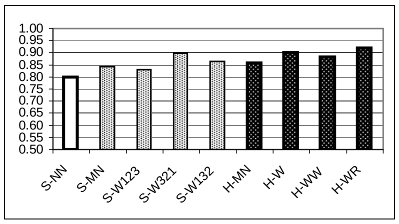
As explained in Section 2.2, the performance of a join algorithm is measured in terms of recall and precision. For comparing the different algorithms, we compute for each algorithm a *rating* which is the harmonic mean of the recall and the precision.

Before presenting the test results, we provide notations for the join algorithms. The sequential nearest-neighbor join is denoted S-NN. The sequential mutually nearest and the holistic mutually nearest are denoted S-MN and H-MN, in correspondence. The sequential normalized-weights method is denoted S-Wijk, when source i and source j are joined first and the result is joined with source k. As for the holistic normalized-weights method—with basic normalization, it is denoted H-W; with approximated normalization, it is denoted H-WW; with normalization according to the real numbers of pairs and singletons, it is denoted H-WR.

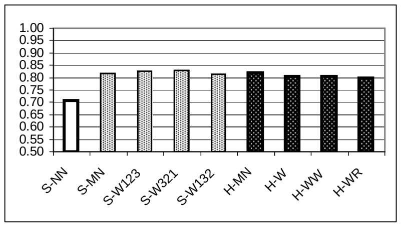
We tested our join algorithms on trios of datasets. The parameters for each trio are the sizes of the datasets and their errors. Since the area and the number of entities are fixed, an increase in the size of a dataset cause an increase in the density and in the overlap. An increase in the error of one dataset causes an increase in the choice factor and, so, for the objects of the other two datasets, there are potentially more corresponding objects in this dataset. In order to examine the effect of changing the sizes and errors of the datasets, we performed four sets of tests.

In the first set, all three datasets have the same size and the same error of $m = 30$ meters. The result of these tests are presented in Figure 3. The graph in Figure 3(a) shows a join of three datasets, each containing 40 objects (small overlap); the graph in Figure 3(b) presents a join of datasets of size 120 (medium overlap); and Figure 3(c) shows a join where all the entities are represented in all the datasets (complete overlap). In the graphs, the Y axis shows the rating of the algorithms. Note that results of sequential algorithms are shown by gray bars, and results of holistic algorithms are shown by black bars.

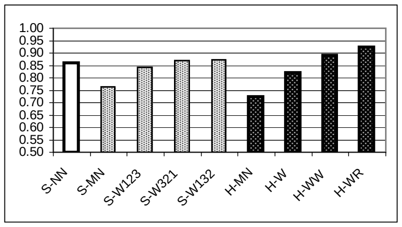
The second set of tests is when all three datasets have the same size, but the errors are different. The results of these tests are presented in Figure 4. As in the previous case, the tests were with a small overlap (Figure 4(a)), a medium overlap (Figure 4(b)) and a complete overlap (Figure 4(c)). In the test, sources 1, 2 and 3 had errors of size 20, 30 and 40, in correspondence.



(a) 40 objects in each dataset.

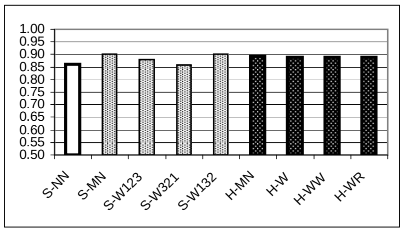


(b) 120 objects in each dataset.

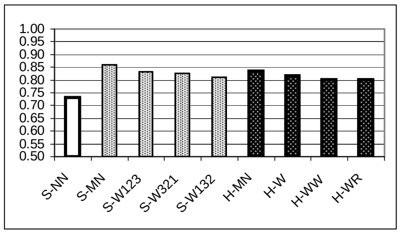


(c) 200 objects in each dataset.

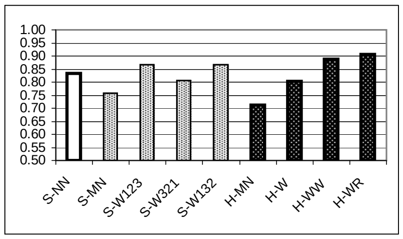
Figure 3: A join of datasets that have the same size and the same error.



(a) 40 objects in each dataset.

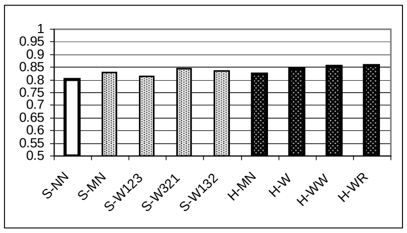


(b) 120 objects in each dataset.

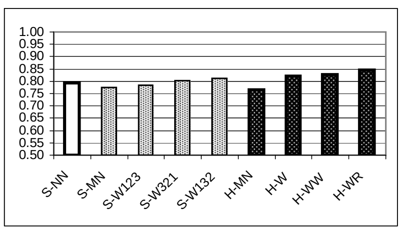


(c) 200 objects in each dataset.

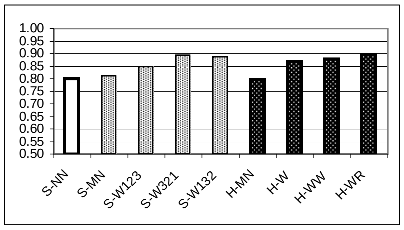
Figure 4: A join of datasets that have the same size and different errors.



(a) All the datasets have the same error.



(b) When the size increases, the error increases.



(c) When the size increases, the error decreases.

Figure 5: A join of datasets with different sizes.

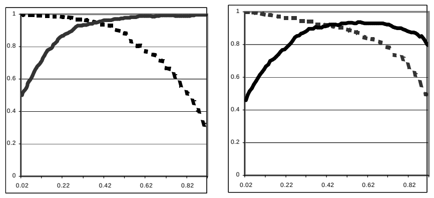
The third set of tests is with datasets that have the same error but different sizes. Figure 5(a) shows the results. In these tests, the error is $m = 30$ meters. The sizes of sources 1, 2 and 3 are 100, 200 and 300 objects, respectively.

Finally, we tested the algorithms with three datasets that have different sizes and different errors. The results are shown in Figure 5(b) and in Figure 5(c), where sources 1, 2 and 3 have 100, 200 and 300 objects, in correspondence. Figure 5(b) presents tests where $m_1 = 20$, $m_2 = 30$ and $m_3 = 40$. That is, when the size of the dataset increases, the size of the error increases. In Figure 5(c), we present the case where $m_1 = 40$, $m_2 = 30$ and $m_3 = 20$, i.e., when the size of the dataset increases, the size of the error decreases.

5.1.3 Conclusions from the Tests

In all our tests, H-WR provided results that are either the best or near the best. But, as explained earlier, using H-WR is limited to cases where the numbers of correct singletons and pairs are known. The method H-WW provided results that are almost as good as the results of H-WR and it can be applied without knowing these numbers.

The holistic mutually nearest join (H-MN) and the sequential mutually-nearest join (S-MN) provide faulty results when the density of the sources is high. The reason for this behavior is that when the sources have low or medium density, there are only a few (or no) pairs of near objects that are not corresponding objects. Thus, in this case, creating join



(a) Holistic. (b) Sequential.

Figure 6: Precision (solid line) and recall (dashed line) as a function of the threshold.

sets from sets of objects that are pairwise mutually nearest is a good strategy. When the sources have high density, there are many pairs of mutually-nearest objects that are not corresponding objects. Hence, in this case, the results of H-MN and S-MN are poor.

The holistic normalized-weights, in its basic version (H-W), is only good when the overlap between the sources is low. We now explain why this happens. In the matching matrix, any possible set has a confidence value, including 2-sets (i.e., sets with two objects) and singletons. When the overlap is medium or small, indeed there are correct join sets of size 1 or 2. But when the overlap is high, there are almost no join sets of size less than 3. Without normalizing to the number of singletons and to the number of 2-sets, many correct 3-sets receive a low confidence value. These sets are discarded in the phase of removing sets that do not exceed the threshold. Thus, the rating of this method is low when the overlap between the sources is high.

As for the sequential normalized-weights method S-Wijk, in each test it provides a good result for at least one of the three join sequences. The difficulty in using this method is that there is no single sequence that always produces the best result. Finding a good sequence can be done using the following rule of thumb. The join should start with the pair of datasets that have the largest overlap and the largest accuracy. This rule, however, does not cover cases when there is a conflict between the degree of the overlap and the degree of accuracy, i.e., cases where the pair with the largest overlap has relatively low accuracy.

The nearest-neighbor join (S-NN) is available in many commercial systems. Nonetheless, our experiments show that for datasets with low or medium overlap, its performance is poor in comparison to all the other methods. The cause for the poor performance is that in a nearest-neighbor join, when a dataset A is joined to a dataset B, every object of A is joined with its nearest neighbor in B (unless there is no object of B within a distance smaller than the mutual-error bound). If the overlap between A and B is medium or low, then many objects of A do not have corresponding objects in B and, thus, many incorrect join sets are produced.

Our tests show that the sequential normalized-weights method (S-W) and the holistic normalized-weights method (H-WR, H-WW) provide the best combination of recall and precision, in most cases. But there are cases where high precision is more important than an optimal combination of recall and precision. In Figure 6, we present typical graphs of the recall and precision as a function of the threshold. These graphs are for the holistic and sequential normalized-weights methods. They show that in the holistic approach, we can increase the precision by increasing the threshold. In the sequential approach, however, precision never reaches 1.

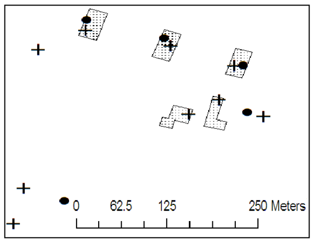


Figure 7: Three datasets of hotels (fragment).

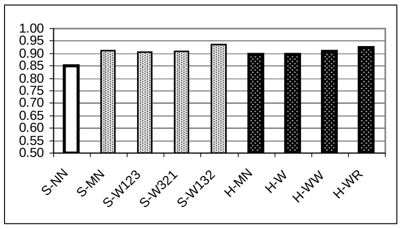


Figure 8: Results of test on real-world data.

5.2 Tests on Real-World Datasets

We tested the different methods on real-world datasets that describe hotels in Tel-Aviv. One dataset, called MUNI, was extracted from a photo of scale 1:7,000 (equivalent to digital maps at the scale of 1:500) and was made by the City of Tel-Aviv. A second dataset, called SOI (Survey of Israel), was extracted from a photo of scale 1:40,000 (equivalent to digital maps at the scale of 1:5,000–1:10,000). A third dataset, called MAPA, was extracted directly from a digital map at the scale of 1:25,000 and was made by the Mapa Corp.

The estimated area of a hotel is 40 × 40 meters. In SOI, hotels are represented by polygons, and we took the center of mass of each polygon as its point location. In MUNI and MAPA, location are given as points, but it is not known which point in the area of each hotel was taken as the location. Hence, we assumed that $\sigma = 20$ (i.e., $m = 50$ meters) in all three datasets.

The MUNI dataset has 73 objects, the SOI dataset has 18 objects and the MAPA dataset has 28 objects. A total of 79 real-world entities are represented in these datasets. A fragment of these datasets is depicted in Figure 7.

The results of the test are presented in Figure 8. As shown, all our methods provided a higher recall-precision rating than the sequential nearest-neighbor join.

5.3 Applicability

Integration of large geographic databases can be carried out in several steps. First, in many cases, spatial data is divided into layers. In such cases, each layer can be integrated separately. Second, in each layer, the integration can be done in one small area at a time (e.g., a municipal region), because geo-spatial datasets typically do not cover large continuous areas. Each small area is expected to include no more than a few hundred objects.

In our tests, when joining three datasets of two hundred objects each, the methods S-NN, S-MN and H-MN completed the computation in about three seconds. For SWijk, about 36 seconds were required. The holistic methods H-W, H-WW and H-WR completed the task in approximately 2.5 minutes.

Since efficiency was not the focus of this work, our algorithms were implemented without any efficiency considerations. However, with simple techniques, such as using an efficient sparse-matrix implementation, it is possible to significantly decrease the computation time.

6. RELATED WORK AND CONCLUSION

The need for integration of heterogeneous data sources arises in many different cases. One example is interoperability of information systems [5, 10, 14]. Another example is mediator systems [2, 21, 22].

For geo-spatial information systems, the data integration problem has two important sub-problems. In *map conflation*, two digital maps are integrated to produce a new map [4, 6, 7, 15, 16, 17]. In *data fusion*, raster data, which is received from sensors, is processed, by means of image-processing techniques, and then integrated [11]. Algorithms for discovering corresponding objects can be part of the solution to these problems.

In this work, we have investigated location-based join of three geo-spatial datasets. Two join approaches, namely, the sequential and the holistic approaches, are presented and compared. The novelty of our work is in developing, for each of the two approaches, effective join algorithms that use only locations of objects.

We showed that the sequential normalized-weights method is effective, that is, the result has a high recall-precision combination when the "right" order of joins is being applied. In the "right" order, we join first the pair of datasets that have the largest overlap and the smallest errors. For the holistic approach, we presented several novel methods. One version of the holistic normalized-weights method provides high recall and precision, under all circumstances.

Comparing the two approaches, the time complexity and the space complexity of the sequential normalized-weights method are lower than those of the holistic normalized-weights method. The holistic approach, however, is capable of providing higher precision than the sequential approach (at the cost of lower recall). Another advantage of the holistic approach is that each join set is given a confidence value. So, if additional information is provided, we can combine this information with the result of the location-based join.

All the methods that are proposed in this paper can be easily applied to a join of any number of sources. In the sequential approach, we simply extend the sequence of joins according to the given number of sources. In the holistic normalized-weights version for n sources, we need to create and normalize an n-dimensional matrix using the same principles as in the case of the 3-dimensional matrix.

Several problems remain for future work. One problem is to optimize the runtime of our algorithms. This is particularly important if we want our algorithms to be included in real-time applications. A second problem is how to utilize most effectively locations that are given as polygons or lines, rather than just points. A third research direction is to combine our approach with other approaches, such as the feature-based approach of [17], topological similarity (e.g., [3]) or ontologies (e.g., [8, 9, 20]).

7. REFERENCES

[1] C. Beeri, Y. Kanza, E. Safra, and Y. Sagiv. Object fusion in geographic information systems. In *Proc. of the 13th International Conference on Very Large Data Bases*, Toronto (Ontario, Canada), 2004.

[2] O. Boucelma, M. Essid, and Z. Lacroix. A WFS-based mediation system for GIS interoperability. In *Proc. of the 10th ACM International Symposium on Advances in Geographic Information Systems*, McLean, (Virginia, US), 2002.

[3] T. Bruns and M. Egenhofer. Similarity of spatial scenes. In *Proc. of the 7th International Symposium on Spatial Data Handling*, Delft (Netherlands), 1996.

[4] 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.

[5] T. Devogele, C. Parent, and S. Spaccapietra. On spatial database integration. *International Journal of Geographic Information Systems*, 12(4), 1998.

[6] 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.

[7] Y. Doytsher, S. Filin, and E. Ezra. Transformation of datasets in a linear-based map conflation framework. *Surveying and Land Information Systems*, 61(3):159–169, 2001.

[8] F. T. Fonseca and M. J. Egenhofer. Ontology-driven geographic information systems. In *Proc. of the 7th ACM International Symposium on Advances in Geographic Information Systems*, Kansas City (Missouri, US), 1999.

[9] F. T. Fonseca, M. J. Egenhofer, and P. Agouris. Using ontologies for integrated geographic information systems. *Transactions in GIS*, 6(3), 2002.

[10] R. Laurini, K. Yetongnon, and D. Benslimane. Gis interoperability, from problems to solutions. In *Encyclopedia of of Life Support Systems (EOLSS)*. Eolss Publishers, 2002.

[11] H. Mayer. Automatic object extraction from aerial imagery—a survey focusing on buildings. *Computer Vision and Image Understanding*, 74(2), 1999.

[12] E. M. Mikhail. *Observations and Least Squares*. University Press of America, 1976.

[13] M. Minami. *Using ArcMap*. Environmental Systems Research Institute, Inc., 2000.

[14] C. Parent and S. Spaccapietra. Database integration: The key to data interoperability. In *Advances in Object-Oriented Data Modeling*. MIT Press, 2000.

[15] B. Rosen and A. Saalfeld. Match criteria for automatic alignment. In *Proc. of 7th International Symposium on Computer-Assisted Cartography (Auto-Carto 7)*, 1985.

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

[17] A. Samal, S. Seth, and K. Cueto. A feature based approach to conflation of geospatial sources. *International Journal of Geographical Information Science*, 18(00):1–31, 2004.

[18] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics*, 35(2):876–879, 1964.

[19] R. Sinkhorn. Diagonal equivalence to matrices with perscribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967.

[20] H. Uitermark, P. V. Oosterom, N. Mars, and M. Molenaar. Ontology-based geographic data set integration. In *Proc. of Workshop on Spatio-Temporal Database Management*, Edinburgh (Scotland), 1999.

[21] G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.

[22] G. Wiederhold. Mediation to deal with heterogeneous data sources. In *Proc. of 2nd International Conference on Introperating Geographic Information Systems*, Zurich (Swizerland), 1999.