Detecting Duplicates in Geoinformatics: from Intervals and Fuzzy Numbers to General Multi-D Uncertainty

Scott A. Starks
Department of Electrical and Computer Engineering
University of Texas at El Paso
El Paso, TX 79968, USA
sstarks@utep.edu

Luc Longpré
Roberto Araiza
Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
{longpre,raraiza,vladik}@utep.edu

Hung T. Nguyen
Department of Mathematical Sciences
New Mexico State University
Las Cruces, NM 88003, USA
hunguyen@nmsu.edu

***Abstract*—Geospatial databases generally consist of measurements related to points (or pixels in the case of raster data), lines, and polygons. In recent years, the size and complexity of these databases have increased significantly and they often contain duplicate records, i.e., two or more close records representing the same measurement result. In this paper, we address the problem of detecting duplicates in a database consisting of point measurements. As a test case, we use a database of measurements of anomalies in the Earth's gravity field that we have compiled.**

In our previous papers [4], [40], we have proposed a new fast ($O(n \cdot \log(n))$) duplication deletion algorithm for the case when closeness of two points (x_1, y_1) and (x_2, y_2) is described as closeness of both coordinates. In this paper, we extend this algorithm to the case when closeness is described by an arbitrary metric.

Both algorithms have been successfully applied to gravity databases.

I. Case Study: Geoinformatics Motivation for the Problem

Geospatial databases: general description. In many application areas, researchers and practitioners have collected a large amount of geospatial data. For example, geophysicists measure values d of the gravity and magnetic fields, elevation, and reflectivity of electromagnetic energy for a broad range of wavelengths (visible, infrared, and radar) at different geographical points (x, y); see, e.g., [36]. Each type of data is usually stored in a large geospatial database that contains corresponding records (x_i, y_i, d_i). Based on these measurements, geophysicists generate maps and images and derive geophysical models that fit these measurements.

Gravity measurements: case study. In particular, gravity measurements are one of the most important sources of information about subsurface structure and physical conditions. There are two reasons for this importance. First, in contrast to more widely used geophysical data like remote sensing images, that mainly reflect the conditions of the Earth's surface, gravitation comes from the whole Earth (e.g., [19], [20]). Thus gravity data contain valuable information about much deeper geophysical structures. Second, in contrast to many types of geophysical data, which usually cover a reasonably local area, gravity measurements cover broad areas and thus provide important regional information.

The accumulated gravity measurement data are stored at several research centers around the world. One of these data storage centers is located at the University of Texas at El Paso (UTEP). This center contains gravity measurements collected throughout the United States and Mexico and parts of Africa.

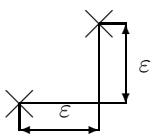
The geophysical use of gravity database compiled at UTEP is illustrated for a variety of scales in [1], [6], [13], [18], [22], [23], [33], [37], [39].

Duplicates: where they come from. One of the main problems with the existing geospatial databases is that they are known to contain many duplicate points (e.g., [16], [29], [35]). The main reason why geospatial databases contain duplicates is that the databases are rarely formed completely "from scratch", and instead are built by combining measurements from numerous sources. Since some measurements are represented in the data from several of the sources, we get duplicate records.

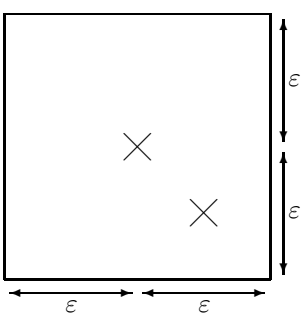
Why duplicates are a problem. Duplicate values can corrupt the results of statistical data processing and analysis. For example, when instead of a single (actual) measurement result, we see several measurement results confirming each other, and we may get an erroneous impression that this measurement result is more reliable than it actually is. Detecting and eliminating duplicates is therefore an important part of assuring and improving the quality of geospatial data, as recommended by the US Federal Standard [12].

Simplest case: duplicates corresponding to interval uncertainty. In the ideal case, when measurement results are simply stored in their original form, duplicates are identical records, so they are easy to detect and to delete. In reality, however, different databases may use different formats and

units to store the same data: e.g., the latitude can be stored in degrees (as 32.1345) or in degrees, minutes, and seconds. As a result, when a record (x_i, y_i, d_i) is placed in a database, it is transformed into this database's format. When we combine databases, we may need to transform these records into a new format – the format of the resulting database. Each transformation is approximate, so the records representing the same measurement in different formats get transformed into values which correspond to close but not identical points $(x_i, y_i) \neq (x_j, y_j)$. Usually, geophysicists can produce a threshold $\varepsilon > 0$ such that if the points (x_i, y_i) and (x_j, y_j) are ε-close – i.e., if $|x_i - x_j| \leq \varepsilon$ and $|y_i - y_j| \leq \varepsilon$ – then these two points are duplicates.



In other words, if a new point (x_j, y_j) is within a 2D *interval* $[x_i - \varepsilon, x_i + \varepsilon] \times [y_i - \varepsilon, y_i + \varepsilon]$ centered at one of the existing points (x_i, y_i), then this new point is a duplicate:



From the practical viewpoint, it usually does not matter which of the duplicate points we delete. If the two points are duplicates, we should delete one of these two points from the database. Since the difference between the two points is small, it does not matter much which of the two points we delete. In other words, we want to continue deleting duplicates until we arrive at a "duplicate-free" database. There may be several such duplicate-free databases, all we need is one of them.

To be more precise, we say that a subset of the original database is obtained by a *cleaning step* if:

- it is obtained from the original database by selecting one or several different pairs of duplicates and deleting one duplicate from each pair, and
- from each *duplicate chain* $r_i \sim r_j \sim \ldots \sim r_k$, at least one record remains in the database after deletion.

A sequence of cleaning steps after which the resulting subset is duplicate-free (i.e., does not contain any duplicates) is called *deleting duplicates*.

The goal is to produce a (duplicate-free) subset of the original database obtained by deleting duplicates.

Duplicates are not easy to detect and delete. At present, the detection and deletion of duplicates is done mainly "by hand", by a professional geophysicist looking at the raw measurement results (and at the preliminary results of processing these raw data). This manual cleaning is very *time-consuming*. It is therefore necessary to design *automated* methods for detecting duplicates.

If the database was small, we could simply compare every record with every other record. This comparison would require $n(n-1)/2 \sim n^2/2$ steps. Alas, real-life geospatial databases are often large, they may contain up to 10^6 or more records; for such databases, $n^2/2$ steps is too long. We need faster methods for deleting duplicates.

From interval to fuzzy uncertainty. Sometimes, instead of a single threshold value ε, geophysicists provide us with several possible threshold values $\varepsilon_1 < \varepsilon_2 < \ldots < \varepsilon_m$ that correspond to decreasing levels of their certainty:

- if two measurements are within ε_1 from each other, then we are 100% certain that they are duplicates;
- if two measurements are within ε_2 from each other, then with some degree of certainty, we can claim them to be duplicates,
- if two measurements are within ε_2 from each other, then with an even smaller degree of certainty, we can claim them to be duplicates,
- etc.

In this case, we must eliminate *certain* duplicates, and mark *possible* duplicates (about which we are not 100% certain) with the corresponding degree of certainty.

In this case, for each of the coordinates x and y, instead of a single interval $[x_i - \varepsilon, x_i + \varepsilon]$, we have a nested family of intervals $[x_i - \varepsilon_j, x_i + \varepsilon_j]$ corresponding to different degrees of certainty. Such a nested family of intervals is also called a *fuzzy set*, because it turns out to be equivalent to a more traditional definition of fuzzy set [3], [24], [30], [31] (if a traditional fuzzy set is given, then different intervals from the nested family can be viewed as α-cuts corresponding to different levels of uncertainty α).

In these terms, in addition to detecting and deleting duplicates under interval uncertainty, we must also detect and delete them under fuzzy uncertainty.

Comment. In our specific problem of detecting and deleting duplicates in geospatial databases, the only fuzziness that is important to us is the simple fuzziness of the threshold, when the threshold is a fuzzy number – or, equivalently, when we have several different threshold values corresponding to different levels of certainty.

It is worth mentioning that in other important geospatial applications, other – more sophisticated – fuzzy models and algorithms turned out to be very useful. There are numerous papers on this topic, let us just give a few relevant examples:

- fuzzy numbers can be used to describe the uncertainty of measurement results, e.g., the results of measuring elevation; in this case, we face an interesting (and technically difficult) interpolation problem of reconstructing the (fuzzy) surface from individual fuzzy measurement results; see, e.g., [28], [34];

- fuzzy sets are much more adequate than crisp sets in describing geographic entities such as biological species habitats, forest regions, etc.; see, e.g., [14], [15];
- fuzzy sets are also useful in describing to what extent the results of data processing are sensitive to the uncertainties in raw data; see, e.g., [26], [27].

What we did in our previous work. In [4], [40], we considered the case when ε-closeness of two points (x_i, y_i) and (x_j, y_j) is described as ε-closeness of both coordinates. In this definition, the set of all points which are ε-close to a given point is a box. For this case, we described an efficient algorithm for detecting and deleting outliers.

New result. In this paper, we extend this algorithm to the case when ε-closeness is described by an arbitrary metric: e.g., Euclidean metric

$$d((x_i, y_i), (x_j, y_j)) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

or l^p-metric

$$d((x_i, y_i), (x_j, y_j)) = \sqrt[p]{|x_i - x_j|^p + |y_i - y_j|^p}.$$

II. Geospatial Databases: Brief Introduction

Geospatial databases: formal description. In accordance with our description, a *geospatial database* can be described as a finite set of *records* $r_1, \ldots, r_n$, each of which is a triple $r_i = (x_i, y_i, d_i)$ consisting of two rational numbers x_i and y_i that describe coordinates and some additional data d_i.

The need for sorting. One of the main objectives of a geospatial database is to make it easy to find the information corresponding to a given geographical area. In other words, we must be able, given one or two coordinates (x and/or y) of a geographical point (center of the area of interest), to easily find the data corresponding to this point and its vicinity.

It is well known that if the records in a database are not sorted by a parameter a, then in order to find a record with a given value of a, there is no faster way than linear (exhaustive) search, in which we check the records one by one until we find the desired one. In the worst case, linear search requires searching over all n records; on average, we need to search through $n/2$ records. For a large database with thousands and millions of records, this takes too much time.

To speed up the search, it is therefore desirable to *sort* the records by the values of a, i.e., to reorder the records in such a way that the corresponding values of a are increasing: $a_1 \le a_2 \le \ldots \le a_n$.

Once the records are sorted, instead of the time-consuming linear search, we can use a much faster *binary search* (also known as *bisection*). At each step of the binary search, we have an interval $a_l \le a \le a_u$. We start with $l = 1$ and $u = n$. On each step, we take a midpoint $m = \lfloor (l + u)/2 \rfloor$ and check whether $a < a_m$. If $a < a_m$, then we have a new half-size interval $[a_l, a_{m-1}]$; otherwise, we have a half-size interval $[a_m, a_u]$ containing a. In $\log_2(n)$ steps, we can thus locates the record corresponding to the desired value of a.

How to sort: mergesort algorithm. Sorting can be done, e.g., by *mergesort* – an asymptotically optimal sorting algorithm that sorts in $O(n \cdot \log(n))$ computational steps (see, e.g., [7]).

III. The Problem of Deleting Duplicates: A Formal Description

Formally, we say that a subset of the database is obtained by a *cleaning step* if:

- it is obtained from the original database by selecting one or several different pairs of duplicates $r_i \sim r_j$ and deleting one duplicate from each pair, and
- from each *duplicate chain* $r_i \sim r_j \sim \ldots \sim r_k$, at least record remains in the database after deletion.

A sequence of cleaning steps after which the resulting subset is duplicate-free (i.e., does not contain any duplicates) is called *deleting duplicates*.

The goal is to produce a (duplicate-free) subset of the original database obtained by deleting duplicates – and to produce it sorted by x_i.

IV. Ideal Case of No Uncertainty

To come up with a general algorithm for detecting and eliminating duplicates under uncertainty, let us first consider an ideal case when there is no uncertainty, i.e., when duplicate records $r_i = (x_i, y_i, d_i)$ and $r_j = (x_j, y_j, d_j)$ mean that the corresponding coordinates are equal: $x_i = x_j$ and $y_i = y_j$.

In this case, to eliminate duplicates, we can do the following. We first sort the records in lexicographic order, so that r_i goes before r_j if either $x_i < x_j$, or ($x_i = x_j$ and $y_i \le y_j$). In this order, duplicates are next to each other.

So, we first compare r_1 with r_2. If coordinates in r_2 are identical to coordinates in r_1, we eliminate r_2 as a duplicate, and compare r_1 with r_3, etc. After the next element is no longer a duplicate, we take the next record after r_1 and do the same for it, etc.

After each comparison, we either eliminate a record as a duplicate, or move to a next record. Since we only have n records in the original database, we can move only n steps to the right, and we can eliminate no more than n records. Thus, totally, we need no more than $2n$ comparison steps to complete our procedure.

Since $2n$ is asymptotically smaller than the time $O(n \cdot \log(n))$ needed to sort the record, the total time for sorting and deleting duplicates is $O(n \cdot \log(n)) + 2n = O(n \cdot \log(n))$. Since we want a sorted database as a result, and sorting requires at least $O(n \cdot \log(n))$ steps, this algorithm is asymptotically optimal.

Comment. It is important to mention that this process does not have to be sequential: if we have several processors, then we can eliminate records in parallel, we just need to make sure that if two record are duplicates, e.g., $r_1 = r_2$, then when one processor eliminates r_1 the other one does not eliminate r_2.

V. The Problem of Deleting Duplicates: Multi-Dimensional Case

Formulation of the problem. At present, the most important case of duplicate detection is a 2-D case, when record are 2-dimensional, i.e., of the type $r = (x, y, d)$. What if we have multi-D records of the type $r = (x, \ldots, y, d)$, and we define $r_i = (x_i, \ldots, y_i, d_i)$ and $r_j = (x_j, \ldots, y_j, d_j)$ to be duplicates if $|x_i - x_j| \le \varepsilon$, ..., and $|y_i - y_j| \le \varepsilon$? For example, we may have measurements of geospatial data not only at different locations (x, y), but also at different depths z within each location.

Related problems of computational geometry: intersection of hyper-rectangles. As we have mentioned in [40], the problem of deleting duplicates is closely related to the problem of intersection of hyper-rectangles in computational geometry [5], [8], [10], [11], [17], [32], [38]. By using these algorithms, we can find and delete all the duplicates in time $O(n \cdot \log^{m-1}(n) + k)$, where k is the total number of pairs that are duplicates to each other.

What was known before. In [4], [40], we have show that if ε-closeness is understood as ε-closeness of all coordinates, then, for all possible dimensions m, the duplicate elimination problem can be solved in time $O(n \cdot \log(n))$ – much faster than for the known computational geometry algorithms.

What we do. In this paper, we extend the above result to the case of a general metric in R^m. This general metric can be described as follows.

Definition 1. *By a* metric*, we mean a triple (S, c, C), where $S \subseteq R^m$ is a convex set that contains 0, and $c > 0$ and $C > 0$ are numbers such that S is* symmetric *(i.e., for every point r, we have $r \in S$ if and only if $-r \in S$) and*

$$[-c, c] \times \ldots \times [-c, c] \subseteq S \subseteq [-C, C] \times \ldots \times [-C, C].$$

*We say that points r and r' are ε-*close *if* $\dfrac{r - r'}{\varepsilon} \in S$.

Comments. The property of c means that S contains all points close to 0. For the case of interval uncertainty, S is a cube:

$$S = [-1, 1] \times \ldots \times [-1, 1].$$

Proposition 1. *For every $m \ge 2$ and for every metric, there exists an algorithm that solves the duplicate deletion problem in time $O(n \cdot \log(n))$.*

Proof. This new algorithm starts with a database of records $r_i = (x_i, \ldots, y_i, d_i)$ and a number $\varepsilon > 0$.

Algorithm 1.

1. For each record, compute the indices
$$p_i = \lfloor x_i/(C \cdot \varepsilon) \rfloor, \ldots, q_i = \lfloor y_i/(C \cdot \varepsilon) \rfloor.$$
2. Sort the records in lexicographic order $\le$ by their index vector $\vec{p}_i = (p_i, \ldots, q_i)$. If several records have the same index vector, check whether some are duplicates of one another, and delete the duplicates. As a result, we get an index-lexicographically ordered list of records: $r_{(1)} \le \ldots \le r_{(n_0)}$, where $n_0 \le n$.
3. For i from 1 to n, we compare the record $r_{(i)}$ with its $\le$-following immediate neighbors; if one of the following immediate neighbors is a duplicate to $r_{(i)}$, then we delete this neighbor.

Let us describe this algorithm in more detail. By an *immediate neighbor* to a record r_i with an index vector $(p_i, \ldots, q_i)$, we mean a record r_j for which for the index vector $\vec{p}_j \ne \vec{p}_i$, for each index, $p_j \in \{p_i - 1, p_i, p_i + 1\}$, ..., and $q_j \in \{q_i - 1, q_i, q_i + 1\}$. An immediate neighbor r_j is called $\le$*-following* if $\vec{p}_i \le \vec{p}_j$.

It is easy to check that if two records are duplicates, then indeed their indices can differ by no more than 1, i.e., the differences $\Delta p \stackrel{\text{def}}{=} p_j - p_i, \ldots, \Delta q \stackrel{\text{def}}{=} q_j - q_i$ between the indices can only take values -1, 0, and 1. Since there are 3 possible values of each of 3 differences, we get 3^m possibilities; one of them is $(0, \ldots, 0)$ which corresponds to this same "cell", so each cell has $\le 3^m - 1$ immediate neighbors.

In Part 2, we start with the first record in the cell (i.e., the first record with the given index vector). Then, we check whether the second record within this same cell is the duplicate of the first one; if no, we add the second record; if yes, we delete the second record. For every new record from this cell, we check whether its a duplicate of one of the already added records.

To describe all $\le$-following immediate neighbors, during Step 3, for each i and for each of $\le 3^m - 1$ difference vectors $\vec{d} = (\Delta p, \ldots, \Delta q)$, we keep the index $j(\vec{d}, i)$ of the first record $r_{(j)}$ for which $\vec{p}_{(j)} \ge \vec{p}_{(i)} + \vec{d}$ (here, $\ge$ means lexicographic order). Then:

- If $\vec{p}_{(j)} = \vec{p}_{(i)} + \vec{d}$, then the corresponding record $r_{(j)}$ (and records from the same cell) are indeed $\le$-following immediate neighbors of $r_{(i)}$, so must check whether it is a duplicate.
- If $\vec{p}_{(j)} > \vec{p}_{(i)} + \vec{d}$, then the corresponding record $r_{(j)}$ is not a $\le$-following immediate neighbor of $r_{(i)}$, so no duplicate check is needed.

We start with $j(\vec{d}, 0) = 1$ corresponding to $i = 0$. When we move from i-th iteration to the next $(i+1)$-th iteration, then, since the records $r_{(k)}$ are lexicographically ordered, for each of $\le 3^m - 1$ vectors $\vec{d}$, we have $j(\vec{d}, i+1) \ge j(\vec{d}, i)$. Therefore, to find $j(\vec{d}, i+1)$, it is sufficient to start with $j(\vec{d}, i)$ and add 1 until we get the first record $r_{(j)}$ for which $\vec{p}_{(j)} \ge \vec{p}_{(i+1)} + \vec{d}$.

To complete the proof, we need to show that Algorithm 1 produces the results in time $O(n \cdot \log(n))$. Indeed, Algorithm 1 consists of a sorting (which takes $O(n \cdot \log(n))$ steps), Part 2, and Part 3.

To estimate the time needed for Part 2, let us count how many records we can have in each cell so that no two are duplicates. Around each record $r_i = (x_i, \ldots, y_i)$, we build a

small cube

$$C_i = \left[x_i - \frac{c\cdot\varepsilon}{2}, x_i + \frac{c\cdot\varepsilon}{2}\right] \times \ldots \times \left[y_i - \frac{c\cdot\varepsilon}{2}, y_i + \frac{c\cdot\varepsilon}{2}\right]$$

with a center at r_i, of half-size $(c\cdot\varepsilon)/2$ and volume $v = (c\cdot\varepsilon)^m$. If two such small cubes intersect, i.e., if $C_i \cap C_j \ni r$ for some point r, then for each coordinate of the corresponding points $r_i = (x_1, \ldots, y_i)$, $r_j = (x_j, \ldots, y_j)$, and $r = (x, \ldots, y)$, we have $|x_i - x| \le (c\cdot\varepsilon)/2$ and $|x_j - x| \le (c\cdot\varepsilon)/2$ hence

$$|x_i - x_j| \le |x_i - x| + |x - x_j| \le c\cdot\varepsilon.$$

Thus, $\dfrac{r_i - r_j}{\varepsilon} \in S$ and the records r_i and r_j are duplicates.

So, if no two records r_i and r_j are duplicates, then the corresponding small cubes C_i and C_j do not intersect. All the small cubes C_i lie within the union of the cell of volume $(2C\cdot\varepsilon)^m$ and its $3^m - 1$ immediate neighbors, i.e., within a zone of volume $V = 3^m \cdot (2C\cdot\varepsilon)^m = (6C\cdot\varepsilon)^m$. Within this zone, we can have at most V/v non-intersecting small cubes of volume v. Thus, the number of non-duplicate records is bounded from above by the ratio $V/v = (6C/c)^m$, which is a constant independent on the number of points n. Thus, in Part 2, we compare every record with $\le V/v$ others. So, for this part, we need time $O(n)$.

During Part 3, for each of $\le 3^m - 1$ vectors $\vec{d}$, we move the corresponding index j one by one from 1 to $n_0 \le n$; for each value of the index, we make $\le (V/v)\cdot(V/v) = \text{const}$ comparisons between records in both cells. Thus, for each vector $\vec{d}$, we need $O(n)$ comparisons.

For a given dimension m, there is a fixed number $\le 3^m - 1$ of vectors $\vec{d}$, so we need the total of $\le (3^m - 1)\cdot O(n) = O(n)$ computational steps. Thus, the total running time of Algorithm 1 is $O(n\cdot\log(n)) + O(n) + O(n) = O(n\cdot\log(n))$. The proposition is proven.

Comment. Since our problem requires sorting, we cannot solve it faster than in $O(n\cdot\log(n))$ steps that are needed for sorting [7]. Thus, Algorithm 1 is asymptotically optimal.

VI. Deleting Duplicates Under Fuzzy Uncertainty

As we have mentioned, in some real-life situations, in addition to the threshold ε that guarantees that ε-close data are duplicates, the experts also provide us with additional threshold values $\varepsilon_i > \varepsilon$ for which ε_i-closeness of two data points means that we can only conclude with a certain degree of certainty that one of these data points is a duplicate. The corresponding degree of certainty decreases as the value ε_i increases.

In this case, in addition to deleting records that are absolutely certainly duplicates, it is desirable to mark possible duplicates – so that a professional geophysicist can make the final decision on whether these records are indeed duplicates.

A natural way to do this is as follows:

- First, we use the above algorithm to delete all the certain duplicates (corresponding to ε).
- Then, we use the same algorithm to the remaining records and mark (but not actually delete) all the duplicates corresponding to the next value ε_2. The resulting marked records are duplicates with the degree of confidence corresponding to ε_2.
- After that, we apply the same algorithm with the value ε_3 to all unmarked records, and mark those which the algorithm detects as duplicates with the degree of certainty corresponding to ε_3,
- etc.

In other words, to solve a fuzzy problem, we solve several interval problems corresponding to different levels of uncertainty. It is worth mentioning that this "interval" approach to solving a fuzzy problem is in line with many other algorithms for processing fuzzy data; see, e.g., [3], [24], [30], [31].

VII. Possibility of Parallelization

If we have several processors that can work in parallel, we can speed up computations:

Proposition 2. *If we have at least $n^2/2$ processors, then, if we simply want to delete duplicates (and we do not want sorting), we can delete duplicates in a single step.*

Proof. For n records, we have $n\cdot(n-1)/2$ pairs to compare. We can let each of $\ge n^2/2$ processors handle a different pair, and, if elements of the pair (r_i, r_j) $(i < j)$ turn out to be duplicates, delete one of them – the one with the largest number (i.e., r_j). Thus, we indeed delete all duplicates in a single step. The proposition is proven.

Comment. If we also want sorting, then we need to also spend time $O(\log(n))$ on sorting [21].

If we have fewer than $n^2/2$ processors, we also get a speed up:

Proposition 3. *If we have at least n processors, then we can delete duplicates in $O(\log(n))$ time.*

Proof. Let us show how Algorithm 1 can be implemented in parallel. Its first stage is sorting, and we have already mentioned that we can sort a list in parallel in time $O(\log(n))$.

Then, we assign a processor to each of n points. For each point, we find each of $\le (3^m - 1)$ indices by binary search (it takes $\log(n)$ time), and check whether the corresponding records are duplicates.

As a result, with n processors, we get duplicate elimination in time $O(\log(n))$. The proposition is proven.

Proposition 4. *If we have $p < n$ processors, then we can delete duplicates in $O((n/p)\cdot\log(n) + \log(n))$ time.*

Proof. It is known that we can sort a list in parallel in time $O((n/p)\cdot\log(n) + \log(n))$; see, e.g., [21].

Then, we divide n points between p processors, i.e., we assign, to each of p processors, n/p points. For each of these points, we check whether each of its $\le \text{const}\cdot(3^m - 1)$ $\le$-following immediate neighbors is a duplicate – which takes $O(\log(n))$ time for each of these points. Thus, overall, checking for duplicates is done in time $O((n/p)\cdot\log(n))$.

Hence, the overall time for this algorithm is

$$O((n/p) \cdot \log(n) + \log(n)) + O((n/p) \cdot \log(n)) =$$
$$O((n/p) \cdot \log(n) + \log(n))$$

– the same as for sorting. The proposition is proven.

Comment: relation to computational geometry. Similarly to the sequential multi-dimensional case, we can solve the duplicate deletion problem much faster than a similar problem of listing all duplicate pairs (i.e., equivalently, all pairs of intersecting hyper-rectangles R_i). Indeed, according to [2], [17], even on the plane, such listing requires time $O(\log^2(n) + k)$.

ACKNOWLEDGMENTS

This work was supported in part by NSF grant EAR-0225670, by Texas Department of Transportation grant No. 0-5453, and by the Japan Advanced Institute of Science and Technology (JAIST) International Joint Research Grant 2006-08. We are very thankful to the anonymous referees for the useful comments.

REFERENCES

[1] Adams, D.C., and Keller, G.R., 1996. Precambrian basement geology of the Permian Basin region of West Texas and eastern New Mexico: A geophysical perspective, American Association of Petroleum Geologists Bulletin 80, 410–431.

[2] Akl, S.G., and Lyons, K.A., 1993. Parallel Computational Geometry, Prentice Hall, Englewood Cliffs, New Jersey.

[3] Bojadziev, G., and Bojadziev, M., 1995. Fuzzy Sets, Fuzzy Logic, Applications, World Scientific, Singapore.

[4] Campos, C., Keller, G.R., Kreinovich, V., Longpré, L., Modave, M., Starks, S.A., and Torres, R., 2003. The Use of Fuzzy Measures as a Data Fusion Tool in Geographic Information Systems: Case Study, Proceedings of the 22nd International Conference of the North American Fuzzy Information Processing Society NAFIPS'2003, Chicago, Illinois, July 24–26, 2003, 365–370.

[5] Chazelle, B.M., and Incerpi, J., 1983. Triangulating a polygon by divide-and-conquer, Proc. 21st Allerton Conference on Communications, Control, and Computation, 447–456.

[6] Cordell, L., and Keller, G.R., 1982. Bouguer Gravity Map of the Rio Grande Rift, Colorado, New Mexico, and Texas Geophysical investigations series, U.S. Geological Survey.

[7] Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C., 2001. Introduction to Algorithms, MIT Press, Cambridge, MA, and Mc-Graw Hill Co., N.Y.

[8] de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O., 1997. Computational Geometry: Algorithms and Applications, Springer-Verlag, Berlin-Heidelberg.

[9] Edelsbrunner, E., 1980. Dynamic Data Structures for Orthogonal Intersection Queries, Report F59, Institute für Informationsverarbeitung, Technical University of Graz.

[10] Edelsbrunner, E., 1983. A new approach to rectangle intersections, Part II, Int'l Journal of Computer Mathematics 13, 221–229.

[11] Edelsbrunner, E., and Overmars, M.H., 1985. Batched dynamic solutions to decomposable searching problems, Journal of Algorithms 6, 515–542.

[12] FGDC Federal Geographic Data Committee, 1998. FGDC-STD-001-1998. Content standard for digital geospatial metadata (revised June 1998), Federal Geographic Data Committee, Washington, D.C., http://www.fgdc.gov/metadata/contstan.html

[13] Fliedner, M.M., Ruppert, S.D., Malin, P.E., Park, S.K, Keller, G.R., and Miller, K.C., 1996. Three-dimensional crustal structure of the southern Sierra Nevada from seismic fan profiles and gravity modeling, Geology 24, 367–370.

[14] Fonte, C.C., and Lodwick, W.A., 2001. Modeling and Processing the Positional Uncertainty of Geographical Entities with Fuzzy Sets, Technical Report 176, Center for Computational Mathematics Reports, University of Colorado at Denver, August 2001.

[15] Fonte, C.C., and Lodwick, W.A., 2003. Areas of Fuzzy Geographical Entities, Technical Report 196, Center for Computational Mathematics Reports, University of Colorado at Denver, March 2003.

[16] Goodchild, M., and Gopal, S. (Eds.), 1989. Accuracy of Spatial Databases, Taylor & Francis, London.

[17] Goodman, J.E., and O'Rourke, J., 1997. Handbook of Discrete and Computational Geometry, CRC Press, Boca Raton, Florida.

[18] Grauch, V.J.S., Gillespie, C.L., and Keller, G.R., 1999. Discussion of new gravity maps of the Albuquerque basin, New Mexico Geol. Soc. Guidebook 50, 119–124.

[19] Heiskanen, W.A., and Meinesz, F.A., 1958. The Earth and its gravity field, McGraw-Hill, New York.

[20] Heiskanen, W.A., and Moritz, H., 1967. Physical Geodesy, W.H. Freeman and Company, San Francisco, California.

[21] Jájá, J., 1992. An Introduction to Parallel Algorithms, Addison-Wesley, Reading, MA.

[22] Keller, G.R., 2001. Gravitational Imaging, In: The Encyclopedia of Imaging Science and Technology, John Wiley, New York.

[23] Keller, G.R., et al., 2006. A community effort to construct a gravity database for the United States and an associated Web portal, In: Sinha, A.K. (ed.), Geoinformatics: Data to Knowledge, Geological Society of America Publ., Boulder, Colorado, 21–34.

[24] Klir, G., and Yuan, B., 1995. Fuzzy Sets and Fuzzy Logic: Theory and Applications, Prentice Hall, Upper Saddle River, NJ.

[25] Laszlo, M.J., 1996. Computational Geometry and Computer Graphics in C++, Prentice Hall, Upper Saddle River, New Jersey.

[26] Lodwick, W.A., 1989. Developing Confidence Limits on Errors of Suitability Analyses in Geographic Information Systems. In: Goodchild, M., and Suchi, G. (Eds.), Accuracy of Spatial Databases, Taylor and Francis, London, 69–78.

[27] Lodwick, W.A., Munson, W., and Svoboda, L., 1990. Attribute Error and Sensitivity Analysis of Map Operations in Geographic Information Systems: Suitability Analysis, The International Journal of Geographic Information Systems 4(4), 413–428.

[28] Lodwick, W.A., and Santos, J., 2003. Constructing consistent fuzzy surfaces from fuzzy data, Fuzzy Sets and Systems 135, 259–277.

[29] McCain, M., and William C., 1998. Integrating Quality Assurance into the GIS Project Life Cycle, Proceedings of the 1998 ESRI Users Conference. http://www.dogcreek.com/html/documents.html

[30] Nguyen, H.T., and Kreinovich, V., 1996. Nested Intervals and Sets: Concepts, Relations to Fuzzy Sets, and Applications, In: Kearfott, R.B., et al, Applications of Interval Computations, Kluwer, Dordrecht, 245–290.

[31] Nguyen, H.T., and Walker, E.A., 1999. First Course in Fuzzy Logic, CRC Press, Boca Raton, FL.

[32] Preparata, F.P., and Shamos, M.I., 1989. Computational Geometry: An Introduction, Springer-Verlag, New York.

[33] Rodriguez-Pineda, J.A., Pingitore, N.E., Keller, G.R., and Perez, A., 1999. An integrated gravity and remote sensing assessment of basin structure and hydrologic resources in the Chihuahua City region, Mexico, Engineering and Environ. Geoscience 5, 73–85.

[34] Santos, J., Lodwick, W.A., and Neumaier, A., 2002, A New Approach to Incorporate Uncertainty in Terrain Modeling. In: Egenhofer, M., and Mark, D. (eds.), GIScience 2002, Springer-Verlag Lecture Notes in Computer Science 2478, 291–299.

[35] Scott, L., 1994. Identification of GIS Attribute Error Using Exploratory Data Analysis, Professional Geographer 46(3), 378–386.

[36] Sharma, P., 1997. Environmental and Engineering Geophysics, Cambridge University Press, Cambridge, U.K.

[37] Simiyu, S.M., and Keller, G.R., 1997. An integrated analysis of lithospheric structure across the East African Plateau based on gravity anomalies and recent seismic studies, Tectonophysics 278, 291–313.

[38] Six, H.W., and Wood, D, 1982. Counting and reporting intersections of d-ranges, IEEE Transactions on Computers C-31, 181–187.

[39] Tesha, A.L., Nyblade, A.A., Keller, G.R., and Doser, D.I., 1997. Rift localization in suture-thickened crust: Evidence from Bouguer gravity anomalies in northeastern Tanzania, East Africa, Tectonophysics, 278, 315–328.

[40] Torres, R., Keller, G.R., Kreinovich, V., Longpré, L., and Starks, S.A., 2004. Eliminating Duplicates Under Interval and Fuzzy Uncertainty: An Asymptotically Optimal Algorithm and Its Geospatial Applications, Reliable Computing, 10(5), 401–422.