Noisy Road Network Matching

Yago Diez[1⋆], Mario Lopez[2], and J. Antoni Sellarès[1⋆]

[1] Universitat de Girona, SPAIN,
{ydiez,sellares}@ima.udg.edu
[2] University of Denver, USA,
mlopez@du.edu

Abstract. Let $\mathcal{N}$ and $\mathcal{M}$ be two road networks represented in vector form and covering rectangular areas R and R', respectively, not necessarily parallel to each other, but with $R' \subset R$. We assume that $\mathcal{N}$ and $\mathcal{M}$ use different coordinate systems at (possibly) different, but known scales. Let $\mathcal{B}$ and $\mathcal{A}$ denote sets of "prominent" road points (e.g., intersections) associated with $\mathcal{N}$ and $\mathcal{M}$, respectively. The positions of road points on both sets may contain a certain amount of "noise" due to errors and the finite precision of measurements. We propose an algorithm for determining approximate matches, in terms of the *bottleneck* distance, between $\mathcal{A}$ and a subset $\mathcal{B}'$ of $\mathcal{B}$. We consider the characteristics of the problem in order to achieve a high degree of efficiency. At the same time, so as not to compromise the usability of the algorithm, we keep the complexity required for the data as low as possible. As the algorithm that guarantees to find a possible match is expensive due to the inherent complexity of the problem, we propose a *lossless filtering* algorithm that yields a collection of candidate regions that contain a subset S of $\mathcal{B}$ such that $\mathcal{A}$ *may* match a subset $\mathcal{B}'$ of S. Then we find possible approximate matchings between $\mathcal{A}$ and subsets of S using the matching algorithm. We have implemented the proposed algorithm and report results that show the efficiency of our approach.

1 Introduction

Spatial analysis of GIS data often requires relating data obtained from two or more different sources. The process of aligning the geometry of two data sets, usually with the purpose of transferring the attributes from one set to the other, is an example of this, referred to as conflation. This type of operation may be needed, for instance, in order to relate a TIGER® file [15] of city streets with a digitized legacy map or with a non-georeferenced map developed in-house. Under such circumstances, it is possible that the two data sets are based on two different coordinate systems, one or both of which are not known precisely or at all.

We consider here the problem of matching a vector data set $\mathcal{M}$ to a subset of another vector data set $\mathcal{N}$, where $\mathcal{N}$ and $\mathcal{M}$ are specified using different and unknown coordinate systems at different but known scales (map scale is basic information, usually known, even in legacy maps). Here, $\mathcal{N}$ and $\mathcal{M}$ cover rectangular areas R and R', respectively, with $R' \subset R$, and the goal is to find an affine transformation (a sequence of rotations, scalings, translations) that locates R' within R. The data sets may contain small imprecisions in geometry, where each true point lies within a circle of radius ϵ from its specified coordinates. We refer to this problem as **noisy road network matching** and define it formally at the end of this section. Note that even though the algorithms are described in terms of road network data, they can be easily extended to other vector formats such as coverages.

While there is a growing body of literature for the problem of relating two data sets with the same coordinate system (see [11,12,14], for example) or, equivalently, with different but known coordinate systems, [3] is the only work we are aware of capable of matching data originating from two different and partially unknown coordinate systems. In [3], the authors assume that R and R' are parallel to each other, thus the target transformation consists of scaling and translation only, and cannot accommodate rotations. In our work, we assume that the map scales are known (as is usually the case for most maps), but the rectangular areas may be arbitrarily rotated and translated with respect to each other. Thus, our work and that of [3] tackle different but, in a sense, complementary aspects of a similar problem.

An example of the type of data we are dealing with is illustrated in Figure 1.

In this paper we present an algorithm for the case of Road Network Matching using the results of [5] as a starting point. One of the specific characteristics of this problem is the presence of "additional" information

⋆ Partially supported by the Spanish Ministerio de Educación y Ciencia under grant TIN2007-67982-C02-02.

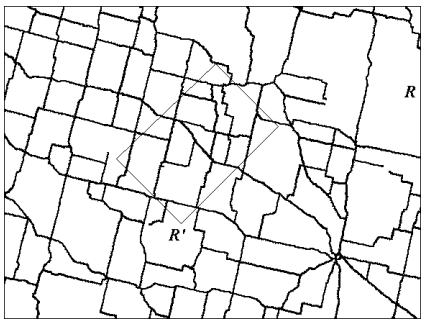

Fig. 1. *A road network $\mathcal{N}$ and subnetwork $\mathcal{M}$ with bounding box R'.*

such as adjacency relationships between intersections or road types. Our algorithm benefits from this auxiliary information. However, in order to keep the algorithm independent of our data sources, we have only used information that can be considered as "evident" to observers.

Our algorithm first examines the line segments of $\mathcal{N}$ and $\mathcal{M}$ in order to extract the endpoints of the input segments. For each such endpoint p, we store a list $Adj(p)$ of its neighbors. For road networks, it is reasonable to assume that the degree of every point is bounded by a small constant, representing the maximum number of roads that may meet at any intersection. This process results in sets $\mathcal{A}$ and $\mathcal{B}$ of *road points* from $\mathcal{M}$ and $\mathcal{N}$ respectively, each of which may either be an *intersection point* (a point of degree bigger than two), a *segment interior point* (a point of degree two) or a *segment endpoint* (a point of degree one). Here, we assume that the shape of the map is well described by the positions of intersection points, as this information tends to be stable across maps of the same region. Accordingly, from now on we will only consider intersections whenever we refer to *(road) points*. Also, since the map scales are known, it is easy to represent both point sets using the same normalized scale. Additionally, we also associate with every road point a list of the type of roads that converge on it (such as highway, presence of tunnel, one-way street, etc.). The road categories are defined in terms of the Census Feature class codes (*CFCC*) present in TIGER/line® files.

We assume that the position of points in both sets $\mathcal{A}$ and $\mathcal{B}$ may contain a certain amount of noise due to the measurement errors or the finite precision of input devices. This is modelled by assuming that the actual position of the points involved may vary by a distance up to a fixed quantity ϵ.

We are now ready to formally define our problem.

Let $\mathcal{A}$ and $\mathcal{B}$ be two road point sets of the same cardinality, expressed using the same normalized scale (i.e., a unit of distance means the same thing in both sets).

An *adjacency-degree preserving bijective mapping* $f : \mathcal{A} \to \mathcal{B}$ maps each point $a \in \mathcal{A}$ to a distinct and unique point $f(a) = b \in \mathcal{B}$ so that $|Adj(a)| = |Adj(b)|$.

Let $\mathcal{F}$ be the set of all adjacency-degree preserving bijective mappings between $\mathcal{A}$ and $\mathcal{B}$. The *bottleneck distance* between $\mathcal{A}$ and $\mathcal{B}$ is defined as:

$$d_b(\mathcal{A}, \mathcal{B}) = \min_{f \in \mathcal{F}} \max_{a \in \mathcal{A}} d(a, f(a)) .$$

The **Noisy Road Network Matching (NRNM)** problem can now be formulated as follows. Given two road point sets $\mathcal{A}$, $\mathcal{B}$, $|\mathcal{A}| = n$, $|\mathcal{B}| = m$, $n \leq m$, and $\epsilon \geq 0$, determine a rigid motion (translations plus rotations) τ for which there exists a subset $\mathcal{B}'$ of $\mathcal{B}$, $|\mathcal{B}'| = n$, such that $d_b(\tau(\mathcal{A}), \mathcal{B}') \leq \epsilon$.

If τ is a solution to the **NRNM** problem, every road point of $\tau(\mathcal{A})$ approximately matches a distinct and unique point of $\mathcal{B}'$ of the same degree, and we say that $\mathcal{A}$ and the subset $\mathcal{B}'$ of $\mathcal{B}$ *approximately match* or are *noisy congruent*.

We now discuss the main differences between our paper and the similar problem described in [3]. We then briefly address the results upon which our approach is based.

– We use the *bottleneck distance* because we believe it is more realistic than the *Hausdorff distance* used in [3], in the context of road network matching (RNM). The reason for this is that Hausdorff distance does not require the points in the two sets to be matched one-to-one. However, for our problem it seems reasonable to require that no two road points in one map be matched to the same road point in the other. This could lead to an input set $\mathcal{A}$ matching a set $\mathcal{B}$ with fewer points, a clearly undesirable situation.

Figure 2 illustrates this problem. A possible set $\mathcal{B}$ (indicated by dots) matches a "smaller" set (indicated by squares) under the Hausdorff distance, but not under bottleneck distance.

Fig. 2. *Two sets of road points that match under Hausdorff distance (for a suitable ϵ) even though they should not match in the NRNM problem.*

- We allow rotations and translations only. Scaling is possible only by assuming that the map scales are known, as is usually the case in practice.
- Another important difference is that the algorithm in [3] computes candidate matchings by finding the transformation T (scaling plus translation) that *exactly* matches a pair of points from $\mathcal{A}$ to a pair from $\mathcal{B}$. It is not hard to find examples where this type of transformation would miss a correct match between $\mathcal{A}$ and a subset of $\mathcal{B}$. For example, Figure 3 contains two sets (one indicated by squares and the other by dots) of three points each. Part (a) of the figure shows that a match is possible (for a suitable ϵ). After computing T using the leftmost two points of each set (illustrated in part (b)), the two leftmost pairs coincide but we have increased the distance for the third pair to a value bigger than ϵ, wrongly concluding that no match can occur. The same happens if we choose any other pair.
- Finally, depending on the values considered for parameter ϵ the practical complexity of the problem changes dramatically. In [3] only very small values of ϵ are considered (the resulting problem is called *Nearly Exact Matching* in the reference). Our algorithm works for any value of ϵ and, in Section 4, we present further discussion of the role of ϵ.

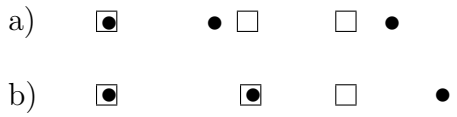

Fig. 3. *a) Three pairs of points, where the pair on the left overlaps and the distances between the two points of each other pair is the same and less than ϵ. The two sets match in the position presented. b) If we force two pairs to overlap (by scaling one of the sets), no match occurs.*

The algorithm presented in this paper uses ideas from Noisy Point Set Matching (**NPSM**) algorithms. The study of this problem was initiated by Alt *et al* [1] who presented an exact $O(n^8)$ time algorithm for finding the transformation that produces the best match between two sets of the same cardinality n. Combining [1] with the techniques proposed by Efrat *et al* [6] the time can be reduced to $O(n^7 \log n)$. Both algorithms suffer from high computational cost due to the inherent complexity of the problem and both use complex data structures that are difficult to implement. The running time is improved here by a lossless filtering preprocess similar to the one proposed in [5] for the problem of colored point set matching. For the choice of practical values of parameter ϵ we also use concepts from pattern matching, as presented in [13].

Finally, as this paper tackles a practical problem, we believe that the evaluation of its efficacy must be done with the greatest rigor. With this goal , we follow the ideas presented in [10] in order to make our experiments as meaningful as possible.

2 NRNM Algorithm

In this section we present a nontrivial adaptation of the algorithm in [5] to the **NRNM** problem. We focus on the aspects that are directly relevant to our problem and only briefly sketch the previous algorithm.

The **NRNM** algorithm consists on two phases: enumeration and testing. The enumeration phase makes the problem finite by partitioning all possible rigid motions of $\mathcal{A}$ into equivalence classes and choosing a

representative motion τ for each class. The testing phase runs a matching algorithm between every set $\tau(\mathcal{A})$ and set $\mathcal{B}$. To speed up calculations during this phase, additional information about road points is considered. The information used should be evident to any observer and, thus, largely independent from the source of the data. Examples of this type of information include the type of road considered and the connectivity of each road point. This will be further discussed in Section 3.

2.1 Enumeration

Generating every possible rigid motion that brings set $\mathcal{A}$ onto a subset of $\mathcal{B}$ is infeasible due to the continuous nature of movement. Following the algorithm presented in [1] and also used in [6,5], we partition the set of all rigid motions into equivalence classes in order to make their handling possible.

This is achieved by realizing that, for any motion that is a solution of the **NRNM** problem, there exists another that is also a solution with the property that two points in $\mathcal{A}$ lie in the boundaries of the noise disks of the two points in $\mathcal{B}$ to which they are matched.

Consequently, $O(n^2m^2)$ 4-tuples of points are considered. For each 4-tuple, we need to work with $O(nm)$ pairs of points in order not to miss any possible matching, obtaining $O(nm)$ *critical events*. The computation of the critical events requires working with high degree algebraic curves known as *coupler curves* of four bar linkages [9]. Summed over all tuples the total number of critical events encountered in the course of the algorithm is $O(n^3m^3)$. For each of these critical events we will need to run the Testing algorithm presented in Section 2.2. The whole NRNM matching algorithm can be summarized as follows:

1. Picture $\mathcal{B}$ in $\mathbb{R}^2$ with a circular noise region surrounding each of its points.
2. Move a rigid copy of set $\mathcal{A}$ over $\mathcal{B}$ until two points in $\mathcal{A}$ hit the boundaries of two of the noise regions.
3. Move set $\mathcal{A}$ without loosing this condition. Every time any other point in $\mathcal{A}$ enters the noise region of any point in $\mathcal{B}$, test for matching using the algorithm in section 2.2.
4. Iterate until all the possible ways in which any two points in $\mathcal{A}$ can hit the boundaries of two of the noise regions of any two points in $\mathcal{B}$ have been tested.

The $O(n^2m^2)$ 4-tuples and $O(n^3m^3)$ critical events express the intrinsic complexity of the problem we are dealing with. They also represent the source of most of the computational effort needed to solve the problem. We will provide experiments to support this statement in Section 4. In general, this cost cannot be decreased as there exist situations where the maximum costs are met. However, in the case of the **NRNM** problems there are ways to reduce it.

Approaches to reduce the cost can be divided into two main types: a) Cutting the computational time needed for every critical event, and b) Reducing the number of 4-tuples and, thus, critical events to be examined. In the first group we find the use of efficient data structures and the "fine tuning" of their implementations. In the second we consider two main strategies. The first one aims at dividing the road points in finer categories and is achieved by considering more information associated to each of them. This information includes adjacency and other evident information, such as *CFCC* codes. These codes are described in the TIGER/lines® technical documentation as "codes that describe the most noticeable characteristics of a feature". More details on this can be found in Section 3.

2.2 Testing

For each 4-tuple we consider every critical event resulting from the enumeration part. These critical events are represented as a series of subintervals of $[0, 2\pi)$. We consider a value ϕ in each of these subintervals and its corresponding motion τ_ϕ. We need to test if there exists a perfect matching between $\tau(\mathcal{A})$ and some subset of $\mathcal{B}$. Whenever the testing part determines a matching of cardinality n we record τ_ϕ and proceed.

We provide Algorithm 1 as a summary of the Enumeration and Testing sections.

The testing algorithm uses ideas from graph theory to look for perfect matchings between $\tau(\mathcal{A})$ and some subset of $\mathcal{B}$. The algorithm uses two main operations that need to be performed efficiently: a) *neighbor* $(D(\mathcal{T}), q)$: for a query point q, use data structure $D(\mathcal{T})$ which stores a point set $\mathcal{T}$, to return a point in $\mathcal{T}$ whose distance to q is at most ϵ, or $\emptyset$, if no such element exists; b) *delete*$(D(\mathcal{T}), s)$: deletes point s from $D(\mathcal{T})$. For our implementation we use the *skip quadtree*, a data structure that combines the best features of a quadtree and a skip list [7].

Algorithm 1 Search for noisy matching $(\mathcal{A}, \mathcal{B})$

{Generate all possible equivalence classes:
ENUMERATION}
for all 4-tuples a_i, a_j, b_k, b_l **do**
  for every couple (a_m, b_p) **do**
    Calculate curve σ_{ijklm}
    $I_{m,p} \leftarrow Intersection((b_p)^\epsilon, \sigma_{ijklm}(x))$
    $\{Critical\_Events\} = \{Critical\_Events\} \cup I_{m,p}$
  end for
end for

{Search for possible matching in every equivalence class: TESTING}
x=0
while $x < 2\Pi$ **do**
  $x \leftarrow$ next critical event
  $\tau \leftarrow$ associated_rigid_motion(x)
  if $(matching(\tau(\mathcal{A}), \mathcal{B}))$ **then**
    {Use algorithm in the "testing" section }
    Record$((\tau))$
  end if
end while

3 Lossless Filtering Preprocess

Most of the computational cost of the **NRNM** algorithm arises from having to consider all possible n^2m^2 quadruples of points described in Section 2.1. Although in some cases it will be necessary to examine all 4-tuples (for example when both sets have the same cardinality), in others there will be subsets of $\mathcal{B}$ that cannot possibly contain matches (for example because they do not contain enough points) and we will be able to avoid looking at the 4-tuples that contain points in these regions. We adapt the lossless filtering preprocess presented in [5]. This preprocess discards subsets of $\mathcal{B}$ according to a series of geometric parameters that are invariant under rigid motion. These parameters help us to describe and compare the shapes of $\mathcal{A}$ and the different subsets of $\mathcal{B}$ that we explore. This ability to discard parts of $\mathcal{B}$ amounts to a pruning of the search space which results in a reduction of the total computational time.

The lossless filtering preprocess yields a collection of *candidate zones* which are regions that contain a subset $\mathcal{S}$ of $\mathcal{B}$ such that $\mathcal{A}$ may approximately match one or more subsets $\mathcal{B}'$ of $\mathcal{S}$. By doing this we transform the initial problem in a number of smaller and independent subproblems and discard those parts of the search space that do not meet the requirements imposed by the geometric parameters. After this preprocessing, we solve the **NRNM** problem between $\mathcal{A}$ and every $\mathcal{S}$ with the algorithm we have already presented.

Figure 3 describes the structure of our solution.

3.1 Lossless Filtering Algorithm

The lossless filtering preprocess consists of two algorithms: quadtree construction and search. The quadtree construction algorithm also consists of two subparts: A compressed quadtree building algorithm that uses the points in $\mathcal{B}$ as sites (without considering their degree), and an algorithm that adds the information related to the geometric parameters being used at each node. The search algorithm traverses the quadtree looking for candidate zones. Below, we provide a more detailed explanation of our solution.

The subdivision of $\mathbb{R}^2$ induced by a certain level of the quadtree consists of axis-parallel squares. At first this does not seem to help because of the requirement to allow set $\mathcal{A}$ to undergo rotations, but we can easily avoid this problem by just searching for a certain axis-parallel square in the quadtree big enough to contain set $\mathcal{A}$ even if it appears rotated. As stated before, we will also require that the square we are looking contains a portion of $\mathcal{B}$ similar to $\mathcal{A}$, in terms of some (rotation invariant) geometric parameters. By doing this, we will be able to temporarily forget about all the possible motions that set $\mathcal{A}$ may undergo and concentrate on those zones of the quadtree where they may actually appear by performing a type of search that is much better suited to the quadtree data structure.

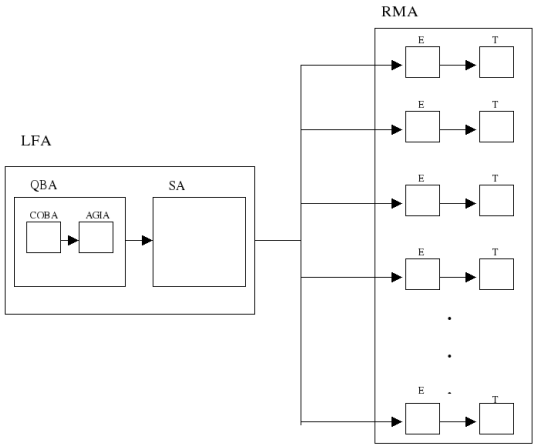

Figure 4: *Overview of all the algorithms used:*

LFA: Lossless Filtering Algorithm
QBA: Quadtree Building Algorithm
CQBA: Compressed Quadtree Building Algorithm
AGIA: Adding Geometric Information Algorithm
SA: Search Algorithm

RMA: Road Matching Algorithm
E: Enumeration
T: Testing

The geometric parameters we use are divided into two main groups:

- **General Parameters**. In this first group we find all parameters that can be considered in point set matching problems where categories are considered. In our cases the categories depend on the degree of adjacency of the (road) points:
  a) parameters based on the fact that we are working with (road) point sets: number of points and histogram of degrees present in the point set.
  b) parameters based on distances between (road) points: maximum and minimum distance between points of every different degree.
- **Road Network specific parameters** c) Histogram of the CFCC codes present in the point set.

For every geometric parameter we will define a *parameter compatibility criterion* that will allow us to discard zones that cannot possibly contain a subset $\mathcal{B}'$ of $\mathcal{B}$ that approximately matches $\mathcal{A}$. See Figure 4 for an example.

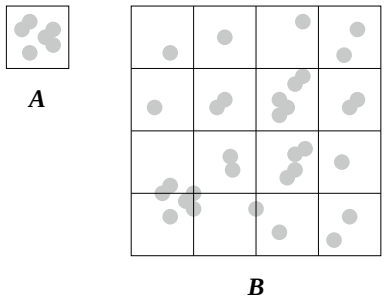

Fig. 4. *There cannot be any $\mathcal{B}'$ that approximately matches $\mathcal{A}$ within the top-left quadrant of B because $\mathcal{A}$ contains six disks (representing points) and that quadrant contains only five.*

We have chosen these parameters, amongst many others, because they are easy to obtain and fairly independent from the data source. In our experiments we have worked mainly with TIGER/lines® data files. These files contain much information that may have been of great help for our algorithm, such as a unique id number for every road intersection point present in the database. Even when this information would have greatly reduced the time of our calculations, one of our goals was to ensure the usability of our algorithm in different situations, so we have restricted our attention to parameters "evident to any observer". With this in mind, we have used the degree of every road point and also the CFCC codes, as they describe the most noticeable characteristics of a feature. Furthermore, both degree and CFCC codes must be present in all records of a TIGER/lines® file.

The cost of adding these geometric parameters to the quadtree of set $\mathcal{B}$ is in $O(m^2)$.

It is important to stress that ours is a conservative algorithm, so we do not so much look for candidate zones as rule out those regions where no candidate zones may appear. A technical issue that arises at this point is that, although our intention was to describe our candidate zones as squares of size s, this will not always be possible, and we will also have to consider groups of two or four squares of size s. Notice that the earlier the discards are made, the bigger the subsets of $\mathcal{B}$ that are discarded.

Search Algorithm

The search algorithm identifies all squares that cover a subset of $\mathcal{B}$ where candidate zones, compatible with $\mathcal{A}$, may be located. This results in three different kinds of candidate zones associated to, respectively one, two or four nodes (see Figure 5) of the tree. The subsets $\mathcal{B}'$ that we are looking for may lie anywhere inside those zones.

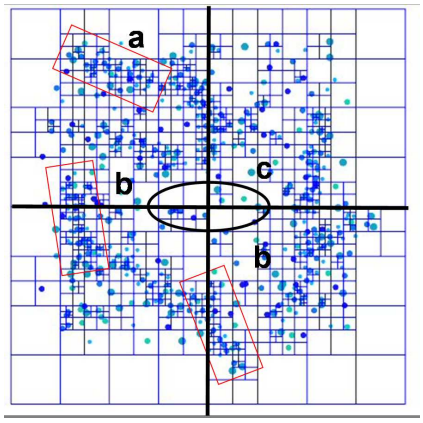

Fig. 5. *Position of the candidate zones in the grid. Overlapping: (a) a single grid-square (corresponding to a single quadtree node), (b) two (vertically or horizontally) neighboring nodes, or (c) four neighboring nodes. In this example we observe occurrences of set $\mathcal{A}$ in zones of the first two types and an ellipse showing where occurrences of the third type (not present) would appear.*

Consequently, the zones considered throughout the search are easily described in terms of the nodes of $\mathcal{Q}_{\mathcal{B}}$ and gradually decrease in size, until they reach s, following the descent of the quadtree. Given that two or four nodes defining a candidate zone need not be in the same branch of $\mathcal{Q}_{\mathcal{B}}$, at some point we may need to explore two or four branches simultaneously. Algorithm 2 outlines the main search function.

The Search algorithm runs in $O(m)$ time. The total cost of the lossless filtering algorithm is $O(m^2)$.

3.2 Overall computational costs

Combining the costs of the lossless filtering preprocess and the Enumerating and Testing algorithms it can be seen that the total cost of the matching algorithm is $O(n^4 m^3 \log m)$. This bound is tight. This shows that from a formal point of view, our process takes, at its worst, the same computational time as the algorithm that does not use the lossless filtering step. Consequently we benefit from any reduction of the computational time that the filtering achieves without any increase in the asymptotic costs.

The cost is high mainly because of the inherent geometric complexity of the problem. In Section 4 we quantify the important improvement in computational costs achieved by using the lossless filtering algorithm, and also by discarding the maximum number of 4-tuples, as described in Section 2.2.

4 Implementation and results

In this section we present experiments that show the degree of efficiency of our algorithms. We have worked with TIGER/lines® files for the counties of Denver (files TGR08031), Adams (TGR08001) and Arapahoe (TGR08005). As we have only used information concerning the position of road points, their connectivity and their associated CFCC codes, it was enough to work with .RT1 files. These files can be found at [15] and the corresponding documentation at [16].

Algorithm 2 Search_1(node N)

```
for all sons S of N do
  if (S is parameter compatible with A ) then
    if ( We have not reached the node size to stop the search) then
      Call Search_1(S)
    else {We have found a candidate node}
      Report candidate zone
    end if
  end if
end for
{Continue in pairs of nodes if necessary (four possibilities)}
for all S1, S2 pairs of neighboring sons of N do
  if (The couple (S1, S2) is parameter compatible with A) then
    if ( We have not reached the node size to stop the search) then
      Call Search_2(S1, S2)
    else {We have found a candidate pair}
      Report candidate zone
    end if
  end if
end for
{Finally, continue in the quartet formed by the four sons if necessary}
(S1, S2, S3, S4): Quartet formed by the sons of N.
if ((S1, S2, S3, S4) are parameter compatible with A ) then
  if ( We have not reached the node size to stop the search) then
    Call Search_4 (S1, S2, S3, S4)
  else {We have found a candidate quartet}
    Report candidate zone
  end if
end if
```

We have implemented all our algorithms in C++ under a Linux environment. We used the g++ compiler without compiler optimizations. All tests were run on a Pentium D machine with a 3 GHz processor.

We consider the set obtained from this data as our set $\mathcal{B}$. To obtain set $\mathcal{A}$, we choose a subset of set $\mathcal{B}$. To choose this subset we randomly choose a point p in $\mathcal{B}$, consider an axis-parallel rectangle that has p lower-left vertex. Then we rotate it a random angle obtaining a set as depicted in Figure 1. Once we have chosen our set $\mathcal{A}$, we apply a random transformation to it that includes:

- Random rotation around one of its points.
- A translation of random vector whose components are bounded by the dimensions of set $\mathcal{B}$.
- A small perturbation applied independently to each of its points in order to simulate noise in data. The modulus of this perturbation is bounded by ϵ.

In most tests, set $\mathcal{B}$ is fixed and sets $\mathcal{A}$ of varying sizes are chosen this way.

The amount of noise ϵ allowed in the data impacts the practical complexity of the problem and, consequently, the computational times obtained. We illustrate this statement with the two extreme cases.

1. $\boldsymbol{\epsilon = 0}$. In this case, the problem would most likely, in practice, not have a solution. As the calculations made to build set $\mathcal{A}$ introduce a certain amount of error (due to rounding, truncation and representation) if we required ϵ to be exactly 0, then there would be no exact match to be found. This behavior can also apply to very small values of ϵ.
2. $\boldsymbol{\epsilon = \infty}$. In this case we consider a value of ϵ too big (meaning, at least bigger than the diameter of set $\mathcal{B}$). In this case, any rigid motion that brings set $\mathcal{A}$ inside a circle of radius ϵ that contains $\mathcal{B}$ would be a solution to our problem. This behavior is clearly undesirable. Such a big value of ϵ stands for knowing very little about the real positions of the points in set $\mathcal{B}$ and any solution found based on this type of data is meaningless.

These two extreme cases show that too small values of ϵ may result in the algorithm missing the solution due to numerical problems, and too big ones may result in meaningless solutions. It is thus very important to consider *reasonable* values of ϵ. To do this, we followed the ideas presented in [13] and studied the empirical values of the *average shortest distance* of set $\mathcal{B}$, $t_{\mathcal{B}}$ defined in the reference as $t_{\mathcal{B}} = \frac{r_{\mathcal{B}}}{2|\mathcal{B}|}$ where $r_{\mathcal{B}}$ is the radius of set $\mathcal{B}$. We have tested different values of ϵ and present results on $\epsilon = t_{\mathcal{B}}$. We also provide some further discussion on this subject on Section 4.4.

4.1 Additional Improvements

In addition to the strategies described until now to reduce the running times of the algorithms (that do so from a theoretical point of view) we have also used some practical strategies that we describe now.

- We observed that the number of road points of degree greater than 5 in our data sets was much smaller than the number of points of degree less than or equal to 5. Taking advantage of this, in every candidate zone we performed an intermediate filtering step looking for a match but considering only road points of degree greater than 5. If this matching was found, we proceeded with the rest of the points and otherwise there was no need to keep on searching in that zone.
- When working with a particular 4-tuple of points, all remaining points in $\mathcal{A}$ and $\mathcal{S}$ have to be tested to decide if they may be matched. This decision is affirmative if one of the points lies in the interior of a coupler curve described by the other. This involves complex calculations, but points that are "too far away" from, for example, the point that we take as the initial point of the coupler curve will never be matched. To reduce the number of pairs considered, we approximated the diameter of the coupler curves by sampling some points on them and calculating their discrete diameter. This resulted in avoiding calculations for most of the points, improving the techniques presented in [5].

Finally, we have to consider the fact that, as we are only looking for one possible match, the computational costs of our experiments also depend on how "early" the algorithm visits the 4-tuple that leads to the solution. To minimize the effect of this, in those discussions where the time spent by the algorithm is the main subject (as is the case of Section 4.2) we will present average times (spent to find a number of sets $\mathcal{A}$ of the same size, chosen randomly) instead of results of just one experiment.

4.2 Effects of the Lossless Filtering algorithm

The performance of the algorithm depends on the effectiveness of the lossless Filtering step and the parameters chosen but, in the worst case, it meets the best (theoretical) running time up to date.

Focusing in our observations of our experiments we can say that in the best case, the initial problem is transformed into a series of subproblems with cardinality n' close to $n = |\mathcal{A}|$, producing a great saving of computational effort. To be fair, we also have to mention that in the worst case, when both sets have similar cardinality, filtering doesn't help. In fact, using the lossless filtering preprocessing step may even increase the running time. In this section we try to quantify this saving in computational time due to filtering. Figure 6 compares the behavior of the Matching algorithm with and without the lossless filtering algorithm (represented by times T1 and T2 in the figure, both in seconds). We fixed set $\mathcal{B}$ to be a subset of 5000 road intersections of the county of Denver and tried sets $\mathcal{A}$ of different sizes. For every size, we present the mean of the times spent by ten sets of the same size. This lossless filtering algorithm uses all the parameters described earlier.

We must state that the sizes considered here are small given the huge computational costs of the algorithm without lossless filtering. It is clear from the figure that, even when the theoretical computational costs are still high due to the complexity inherent to the problem, using the lossless filtering results in significant computational savings.

4.3 Discussion on geometric parameters

In this section we provide results that measure the effectiveness of the different geometric parameters used during the lossless filtering algorithm. Table 1 presents the number of candidate zones and computational

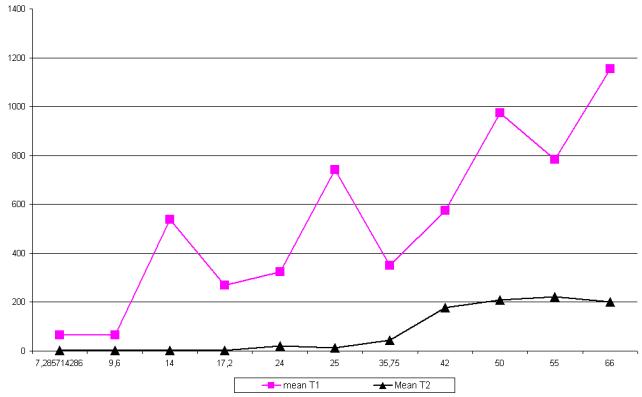

Fig. 6. *Running times not using lossless filtering step (T1) and using it (T2). X axis represents the mean values of the cardinals of sets $\mathcal{A}$ in every test and axis Y represents time in seconds.*

costs for the search algorithm resulting from: 1) using only the "number of points" (Num.) parameter; 2) using (1) plus the histogram of points degrees (Histo.); 3) using (1), (2) plus the "maximum and minimum distance between points of the same degree" (Dist.) parameters; 4) adding to the previous three the CFCC codes associated to each road intersection point.

For the tests we used two main data sets. The first one contains 12052 road intersections in the county of Denver. The second contains, in addition, data about road intersections in the counties of Adams and Arapahoe totaling 39950 road intersections. We run the lossless filtering algorithm for different sets $\mathcal{A}$ of varying sizes. Table 1 presents the number of candidate zones obtained for every filtering criteria and the time consumed in each case by the search algorithm, the time needed to build the quadtree data structure was approximately 16 seconds for the first data set, and approximately 150 seconds for the second.

$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{B}\rvert$	Num.		Num./Histo.		Num./Histo./Dist.		Num./Histo./Dist./CFCC	
		#zones	Time(s)	*#zones*	Time(s)	*#zones*	Time(s)	*#zones*	Time(s)
25	12052	217	<< 0.01	188	<< 0.01	187	<< 0.01	31	<< 0.01
100	12052	194	<< 0.01	61	<< 0.01	52	0.19	52	0.19
250	12052	57	<< 0.01	19	<< 0.01	19	0.016	9	0.17
500	12052	21	<< 0.01	17	<< 0.01	16	<< 0.01	10	<< 0.01
750	12052	20	<< 0.01	15	<< 0.01	14	5.41	8	5.41
1000	12052	19	<< 0.01	10	<< 0.01	8	1.57	6	1.59
2000	39925	52	<< 0.01	16	<< 0.01	12	7.55	6	7.53
5000	39925	8	<< 0.01	4	<< 0.01	4	0.12	2	0.14
10000	39925	12	<< 0.01	4	<< 0.01	4	0.01	2	0.03
12500	39925	11	<< 0.01	4	<< 0.01	4	<< 0.01	2	<< 0.01

Table 1. *Effects of the different geometric parameters on the lossless filtering algorithm.*

We observe that the number of candidate zones is always lower when we use more "elaborate" geometric parameters. The main reduction in the number of candidate zones is usually obtained when the histogram of road intersection degrees is considered, but in most cases this reduction continues when distances between road intersections of the same degree and CFCC codes are considered. We also want to highlight two special cases:

- We noticed that the number of candidate zones obtained for sets of parameters (2) and (3) differ more in the second data set. As this stands for considering the distances of road intersections of the same degree or only their histogram, this may be caused by a more regular distribution of road intersections inside the city of Denver than in the other counties considered.
- In some cases, the number of candidate zones obtained for sets of parameters (3) and (4) is the same. The difference in this case is whether we consider or not CFCC codes. In some of the test cases, all the roads

that formed the intersections of set $\mathcal{A}$ had the same code (A41 which, according to the TIGER/lines® documentation, stands for *Local, neighborhood, and rural road, city street, unseparated*). We studied the distribution of road CFCC codes for both data sets and found that more than 93.5% of the roads in Denver and about 90% on Denver, Adams and Arapahoe fall in this category.

With respect to computational times, the time needed to perform the search algorithm is bigger when we use more geometric parameters, but still much less than the cost of the Matching algorithm. In conclusion, the use of more geometric parameters results in the output of less, and better, candidate zones. Moreover, although the use of more geometric parameters slightly increases computational time, the cost of the lossless filtering algorithm is still much smaller than that of the Matching algorithm.

4.4 Computational Performance

Here we seek to evaluate the performance of the whole algorithm, with all improvements, in some "real life" situation. We ran tests where set $\mathcal{B}$ corresponds to the road intersection points in the county of Denver (cardinality 12052) and used different sets $\mathcal{A}$ chosen randomly inside rectangles of varying sizes. Table 2 shows:

- The cardinalities of the sets involved in the test.
- The average number of candidate zones examined before finding the solution. This value is presented in order to provide an idea on how effective the filtering step is in transforming the initial problem into a series of smaller subproblems with similar cardinalities between $\mathcal{B}'$ and $\mathcal{A}$.
- Finally, we present data on the number of 4-tuples and pairs examined. This data helps us to highlight where the main computational effort is, as it is directly related to the time spent by the algorithm.

$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{B}\rvert$	$\overline{n'}$	#4-tuples	#pairs	Time(s)
10	12052	55	6	93	15.31
50	12052	111	7	218	31.84
75	12052	481	18	1381	73.17
100	12052	653	1	670	27.38
150	12052	1880	4	3484	243.97
200	12052	1089	5	7918	849.65
250	12052	1088	20	19856	2464.71
300	12052	1888	5	2697	404.56

Table 2. *Performance of the complete algorithm.*

Concerning the mean value of n', denoted $\overline{n'}$, we observe that the filtering algorithm manages to take away most of the complexity of the problem. This works better for smaller sets, as the algorithm is able to locate candidate zones built by smaller nodes in the quadtree. We can also observe that the total computational cost of the algorithm is greatly influenced by the number of 4-tuples or pairs examined, as tests for sets of similar sizes may need very different computational effort (this is illustrated by the rows corresponding to $|\mathcal{A}| = 200$ and $|\mathcal{A}| = 250$). Consequently, the main part of the computational effort can be attributed to the complexity inherent to the problem. Finally the computational time increases with $|\mathcal{A}|$, although we believe that it is kept to reasonable levels given the complexity of the calculations involved.

Other values of ϵ

To complement the discussion on the values of ϵ provided at the beginning of this section, we present some tests for different values of ϵ.

As expected, as the value of ϵ decreases, so does the number of 4-tuples that lead to a solution (arriving to zero in the limit case described previously). This results in an increase of the total computational time due to having to "search more" before finding a solution. Also, smaller values of ϵ result in fewer 4-tuples and pairs to be checked, or sometimes allow for better filtering and smaller candidate zones. Both of these considerations result in a decrease of computational time. Which of the two tendencies is more important? Basically it depends on the data set being searched, for most of our study cases, smaller values of ϵ required higher computational time, but in some others (generally for bigger realizations of set $\mathcal{A}$) the opposite held. Table 3 shows examples of the two cases and even one where both tendencies produce similar computational times for both values of ϵ.

ϵ	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{B}\rvert$	$\overline{n'}$	#4-tuples	#couples	Time(s)
$t_{\mathcal{B}}$	10	12052	55	4	25	22.51
$\frac{t_{\mathcal{B}}}{2}$	10	12052	55	1	18	22.48
$t_{\mathcal{B}}$	31	12052	99	162	2892	79.87
$\frac{t_{\mathcal{B}}}{2}$	31	12052	99	1	135	19.23
$t_{\mathcal{B}}$	230	12052	540	13	9192	472.08
$\frac{t_{\mathcal{B}}}{2}$	230	12052	1089	14	22183	2114.26

Table 3. *Compared performance for different values of ϵ.*

5 Conclusions

In this paper we have presented, to the best of our knowledge, the first formalization of the **NRNM** problem in terms of the *bottleneck* distance. We have studied the specific characteristics of the problem in order to obtain an algorithm that takes advantage of them, without compromising its practical value, independent of the data source being used. We have presented theoretical and practical discussions on this algorithm and obtained running times that are fast, in light of the inherent complexity of the problem. Finally, we have presented experiments that have allowed us to highlight the most important aspects of the algorithm as well as to gain insight into important characteristics of the problem we were dealing with.

References

1. H. Alt, K. Mehlhorn, H. Wagener and E. Welzl. Congruence, similarity and symmetries of geometric objects. *Discrete & Computational Geometry*, 3:237–256, 1988.
2. R.P. Brent. Algorithms for Minimization Without Derivatives. *Englewood Cliffs*, NJ: Prentice-Hall, 1973.
3. C. Chen, C. Shahabi, M. Kolahdouzan, and C.A. Knoblock. Automatically and Efficiently Matching Road Networks with Spatial Attributes in Unknown Geometry Systems. *3rd Workshop on Spatio-Temporal Database Management* (STDBM'06), September 2006.
4. C. Chen. Automatically and Accurately Conflating Road Vector Data, Street Maps and Orthoimagery. *PhD Thesis*, 2005.
5. Y.Diez, J.A. Sellarès. Efficient Colored Point Set Matching Under Noise. *ICCSA 2007, Lecture Notes on Computational Science 4705*, 26-40, Springer-Verlag, 2007.
6. A. Efrat, A. Itai and M.J. Katz. Geometry helps in Bottleneck Matching and Related Problems. *Algorithmica*, 31:1–28, 2001.
7. D. Eppstein, M.T. Goodrich, and J.Z. Sun. The skip quadtree: a simple dynamic data structure for multidimensional data. *21st ACM Symposium on Computational Geometry*, 296–305, 2005.
8. J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *S*IAM Journal on Computing, 2(4):225–231, 1973.
9. K.H. Hunt. Kinematic Geometry of Mechanisms (Chapters 4,7), *Oxford University Press*, 1978.
10. D.S. Johnson. A Theoretician's Guide to the Experimental Analysis of Algorithms, *Goldwasser M, Johnson DS, McGeoch CC, editors. Proceedings of the fifth and sixth DIMACS implementation challenges. Providence, RI: American Mathematical Society*, 2002.
11. A. Saalfeld. Conflation: Automated Map Compilation, *International Journal on Geographical Information Systems*, 2(3):217–228, 1988.
12. V. Walter and D. Fritsch. Matching Spatial Data Sets: a Statistical Approach. *International Journal of Geographical Information Science*, 13(5):445–473, 1999
13. P.B. Van Wamelen, Z. Li, S.S. Iyengar, A fast expected time algorithm for the 2-D point pattern matching problem. *Pattern Recognition. 37(8) 2004, 1699–1711.*
14. J.M. Ware and C.B. Jones. Matching and Aligning Features in Overlayed Coverages. *6th ACM International Symposium on Advances in Geographic Information Systems*, 28–33, 1998.
15. U.S. Census Bureau Topologically Integrated Geographic Encoding and Referencing system, TIGER®, TIGER/Line® and TIGER®-Related Products *http://www.census.gov/geo/www/tiger/.*
16. U.S. Census Bureau TIGER/Line® files, Technical documentation. First Edition 2006. *http://www.census.gov/geo/www/tiger/tiger2006fe/ TGR06FE.pdf .*