Accelerating Profile Queries in Elevation Maps

Feng Pan, Wei Wang, Leonard McMillan
University of North Carolina at Chapel Hill
{panfeng, weiwang, mcmillan}@cs.unc.edu

Abstract

Elevation maps are a widely used spatial data representation in geographical information systems (GIS). Paths on elevation maps can be characterized by profiles, which describe relative elevation as a function of distance. In this research, we address the inverse of this mapping — given a profile, how to efficiently find paths that could have generated it. This is called the profile query problem. Profiles have a wide variety of uses that include registering tracking information, or even other maps, to a given map. We describe a probabilistic model to characterize the maximal likelihood that a point lying on a path matches the query profile. Propagation of such probabilities to neighboring points can effectively prune the search space. This model enables us to efficiently answer queries of arbitrary profiles with user-specified error tolerances. When compared to existing spatial index methods, our approach supports more flexible queries with orders of magnitude speedup.

1 Introduction

In geographical information systems (GIS) systems a common problem is to align a given path to its proper position on a map. Paths can be specified in a variety of ways ranging from full geospatial coordinates (longitude, latitude, and reference to a geodetic model) to compass bearings with odometry. While it is simple to generate paths from elevation maps, the inverse problem of finding a compatible embedding for a particular path, is more difficult. We address this problem for a particular form of path specification called a *profile*. A profile assumes only that the relative altitude and distances along the path are known. The goal of a *profile query* is to find a set of potential alignments on an elevation map (paths) which are consistent with a given profile.

Profiles have a wide variety of uses including

- Hydrology studies
- Registering tracking information to a given map
- Estimating true distances travelled
- Road planning and transportation engineering
- Design of road race courses (e.g. marathons)

In a typical Digital Elevation Map (DEM), elevation data is regularly sampled on a uniform grid. Our approach maps a DEM to a graph consisting of segments between their 8 neighboring elevation measurements. In a DEM containing $n \times m$ points, the total number of paths containing k segments would be $O(n \cdot m \cdot 8^k)$. Even for a small 100×100 map, there are $O(10^{10})$ 8-segment paths. This huge search space poses significant challenges for existing spatial index structures and querying algorithms. Current spatial index structures can store trajectories. However, the huge number of paths in an elevation map prevents the spatial index structure from handling the profile-query problem efficiently. Allowing for variable length queries makes the problem even harder for typical spatial index structures.

In this paper, we develop a probabilistic model to solve the profile query problem. Our model characterizes the probability that a given map point matches a query profile. The propagation of joint probabilities to neighboring points effectively prunes the search space. This model enables us to efficiently support queries of arbitrary profiles with user-specified error tolerances.

Our query algorithm operates in two steps. In the first step, the probabilistic model quickly locates possible endpoints of the query. Then, in the second step, the searching focuses on the neighborhoods around these endpoints. We also describe several optimizations to handle large elevation maps efficiently. Experiments show that our approach supports more flexible queries with orders of magnitude better performance than approaches based on existing spatial-indexing methods.

2 Preliminaries

In this section, we discuss the terminology, notation, and assumptions of our approach. We define elevation maps, paths, and elevation profiles, which we will henceforth refer to as simply *profiles*. We will also give a formal specification of the *profile query* problem.

A digital elevation map approximates a heightfield function, $z = h(x, y)$. Each point is characterized by a tuple (x, y, z). A common practice is to sample the elevation map at regular intervals in both x and y. This leads to a sampling lattice, given an elevation map of size $n \times m$, its point set can be represented as tuples $\{(i, j, h(i, j)) | 1 \leq i \leq n, 1 \leq j \leq m, i, j \in N\}$. Furthermore, an elevation map can be represented as a matrix M, where $M_{ij} = h(i, j)$. An example elevation map matrix is shown in Figure 1.

For each point $(i, j, h(i, j))$ of a map, we define its neighborhood points as the eight neighboring points around it.

$$M = \begin{bmatrix} 0.3 & 6.7 & 18.3 & 6.7 & 0.3 \\ 6.7 & 135.3 & 367.9 & 135.3 & 6.7 \\ 18.3 & 367.9 & 1000 & 367.9 & 18.3 \\ 6.7 & 135.3 & 367.9 & 135.3 & 6.7 \\ 0.3 & 6.7 & 18.3 & 6.7 & 0.3 \end{bmatrix}$$

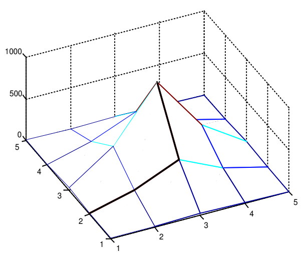


Figure 1. An elevation map of size 5×5

A path starts at a point in the elevation map. From each point, the path can go to any of its eight neighbors. Therefore, a path can be represented by a list of points

$$\begin{gathered} path = \{(x_1, y_1, z_1), ..., (x_n, y_n, z_n)\} \\ |x_i - x_{i-1}| \le 1, |y_i - y_{i-1}| \le 1, x_i, y_i \in N \end{gathered}$$

For example, a path in Figure 1 can be

$$path_1 = \{(1, 2, 6.7), (2, 2, 135.3), (3, 2, 367.9), (3, 3, 1000)\}$$

which is shown as the bold curve.

A profile describes relative elevation as a function of distance. A profile can also be represented as segments on a path characterized by the slope and projected Euclidian distance on xy plane of each segment,

$$\begin{aligned} &profile(path) = \{(s_1, l_1), ...(s_k, l_k)\} \\ &l_i = dis((x_i, y_i), (x_{i+1}, y_{i+1})) \\ &s_i = (z_i - z_{i+1})/l_i \end{aligned}$$

If the xy projected distance is unavailble it can be derived from the geodesic distance, g, as follows:

$$l_i = sqrt(g^2 - (z_i - z_{i+1})^2)$$

A path consisting of n points generates a profile with $n - 1$ segments. We call a profile of size k if it has k segments.

Given a profile of size k, its profile prefixes are:

$$profile^{(i)} = \{(s_1, l_1), ...(s_i, l_i)\}, 1 \le i \le k$$

Where $profile^{(k)} = profile$.

In later discussions, we will denote the query profile as

$$Q = \{(s_1^q, l_1^q), ...(s_k^q, l_k^q)\}$$

We also define two distance measurements between profiles of same size, D_s and D_l, for the slope and projected distance respectively.

$$D_s(profile_u, profile_v) = \sum_{i=1}^{k} |s_i^u - s_i^v|$$

$$D_l(profile_u, profile_v) = \sum_{i=1}^{k} |l_i^u - l_i^v|$$

In order to enable a user-specified error tolerance, we define two error tolerance parameters, δ_s and δ_l, for the two distance measurements respectively.

Problem Definition of Profile Query: Given a query profile, Q, find all the paths in the elevation map that can generate profiles satisfying (1) and (2).

$$D_s(profile, Q) \le \delta_s \quad (1)$$

$$D_l(profile, Q) \le \delta_l \quad (2)$$

A path matches the query profile if it satisfies (1) and (2).

3 Related Work

The profile query problem is closely related to other spatial-indexing and spatial-query problems. Indeed, some previously developed methods support profile queries. However, previous methods tend not to scale well as the map size increases or as the length of the profile grows. Furthermore, these methods do not, in general, enable the specification of error bounds.

Spatial indexing structures like the R-tree [5, 10] or MVR-tree [7, 11] can be used to index paths by mapping them into points in a higher dimensional space. However, the total number of possible paths is an exponential function of the size of the map, even for paths of fixed length. Therefore, directly indexing all possible paths is intractable.

To avoid the explosion in the number of paths, a traditional indexing structure like B+tree can be used to index all the segments in the map according to their length and slope. By querying the profile segment by segment, $B+$ tree can find the best candidates for each segment, and the best matching paths can be generated by concatenating those candidates. However, since *profile query* allows user-specified error tolerance, B+tree can typically find thousands of candidates for each segment of the profile. During the concatenating procedure, the total number of candidate paths generated will be combinatorially larger. Since candidate paths are pruned once they exceed the error tolerance, the procedure has to test a huge number of candidate paths.

A related problem from the field of motion planning is to estimate a robot's position according to the world model (map) based on sensor data. Markov Localization [3, 1, 8] is a probability framework which estimates the possible position of a moving robot conditioned on the sensor data. By considering the query profile as the sensor data, Markov Localization can estimate the end point of a matching path. However, its probability model does not reflect the goodness of a matching path, for example, the end point of a best matching path may not have the highest probability value in Markov Localization. Therefore, it is unable to find matching paths correctly.

Queries over moving object trajectories have been extensively studied in [6, 2, 4]. In these projects, trajectories are considered as spatial-temporal curves and the queries differ from ours. Instead of searching trajectories of specific shape, they want to find trajectories passing objective regions during specific time intervals or following some temporal order. Since the object trajectories are given, spatial index structures are employed and perform well. In contrast, only an elevation map is given in our problem which implicitly contains a huge number of possible paths, exponential to the size of the map. It is infeasible to compute and index all possible paths from an elevation map.

4 Probabilistic Model

In this section, we develop the probabilistic model used to solve the *profile query* problem. The objective is to estimate the probability that a point is at the end of a matching path. The composition of high probability end points leads to a list of path candidates. Let L_k be the corresponding random variable given query profile Q of size k and let p be a point in the map. Our model establishes a posterior distribution over L_k conditioned on the query profile.

$$P(L_k = p|Q) = P(L_k = p|\{(s_1^q, l_1^q), ...(s_k^q, l_k^q)\}) \quad (3)$$

In our algorithm, the probabilistic model acts as a scoring function model which measures the quality of path endpoints according to their similarity with the query profile. Other scoring functions (distance functions) can also be used in place of our algorithm's probabilistic model. We explain the reasoning behind our model in Section 5.

First, we introduce an important model property necessary to solve *profile query* problem.

Property 4.1 *A point having higher $P(L_k = p|Q)$ value corresponds to a better path.*

We will say $path_u$ is better than $path_v$ if their profiles, $profile_u$ and $profile_v$, satisfy the following relation:

$$D_s(profile_u, Q)/b_s + D_l(profile_u, Q)/b_l$$
$$\leq D_s(profile_v, Q)/b_s + D_l(profile_v, Q)/b_l \quad (4)$$

b_s and b_l are normalizing factors used to weight the slope and distance metrics, D_s and D_l. The values of b_s and b_l are set to be proportional to the error tolerances δ_s and δ_l respectively. In our implementation, we set $b_s = 10 \cdot \delta_s$ and $b_l = 10 \cdot \delta_l$.

Equation 4 is a necessary condition for Equations 1 and 2 to be satisfied if $D_s(profile_v, Q) = \delta_s$ and $D_l(profile_v, Q) = \delta_l$. The probabilistic model must satisfy Property 4.1, in order to relate the probability values to the two distance measurements D_s and D_l. By choosing an appropriate function, we can convert the error tolerance of D_s and D_l to an equivalent error tolerance of the probability, which can then be used to prune points in the map.

In order to estimate the maximum likelihood path while considering that paths can only be extended to neighboring points, we use the following equation:

$$P(L_k = p|Q^k)$$
$$= \alpha_k \cdot max_{p'}\{P(L_k = p|(s_k^q, l_k^q), L_{k-1} = p')P(L_{k-1} = p'|Q^{(k-1)})\} \quad (5)$$

where $P(L_{k-1} = p'|Q^{(k-1)})$ is the distribution conditioned on the profile prefix $Q^{(k-1)}$. The constant α_k is used as a normalizing factor to make $\sum_p P(L_k = p|Q) = 1$.

Equation 5 illustrates the propagation of the probabilities. Conditioned on the profile prefix $Q^{(k-1)}$, the model finds the most probable point to extend the current profile, $Q^{(k-1)}$. The profile is extended with the addition of segment (s_k^q, l_k^q), a choice that 1) is adjacent to the previous best endpoint, and 2) maximizes the probability of the profile-path match. We show later in this section that Equation 5 satisfies Property 4.1.

The term $P(L_k = p|(s_k^q, l_k^q), L_{k-1} = p')$ in Equation 5 describes how the probability is propagated. In Equation 5, p' represents all points in the map. They can be divided into two parts, points that are neighbors of p and points that are not neighbors of p. As we defined in Section 2, a path can only be extended to the neighboring points of a current endpoint. Therefore, for the points that are not neighbors of p, we have

$$P(L_k = p|(s_k^q, l_k^q), L_{k-1} = p') = 0,$$
$$|x_{p'} - x_p| > 1 \mid |y_{p'} - y_p| > 1 \quad (6)$$

For those neighbors of p, we assume two independent Laplacian distributions for $P(L_k = p|(s_k^q, l_k^q), L_{k-1} = p')$. Let s be the slope of the segment between p and p' and l be its projected distance on xy plane.

$$P(L_k = p|(s_k^q, l_k^q), L_{k-1} = p')$$
$$= (\frac{1}{2b_s})(\frac{1}{2b_l})e^{-|s-s_k^q|/b_s}e^{-|l-l_k^q|/b_l}, |x_{p'} - x_p| \leq 1 \& |y_{p'} - y_p| \leq 1 \quad (7)$$

Assuming a Laplacian distribution for $P(L_k = p|(s_k^q, l_k^q), L_{k-1} = p')$ assigns higher values if (s, l) is closer to (s_k^q, l_k^q). As a simplification, we will set $b_l = b_s = b$ in the later parts. It is easy to see that the values of b_l and b_s do not affect the key properties of the probabilistic model nor the correctness of our algorithm.

Equation 5 suggests that the probability is propagated through neighboring points, and the final probability of point p corresponds to one of the paths ending at p which can be found by linking p and the maximal p' in each step of recursion. According to Equation 5, we get the relationship between the probability of p and its corresponding path.

$$P(L_k = p|Q)$$
$$= P_0 \cdot (\prod_{i=1}^{k} \alpha_i)(\frac{1}{2b})^{2k} e^{-(\sum_{i=1}^{k} |s_i - s_i^q| + \sum_{i=1}^{k} |l_i - l_i^q|)/b}$$
$$= P_0 \cdot (\prod_{i=1}^{k} \alpha_i)(\frac{1}{2b})^{2k} e^{-(D_s(profile,Q) + D_l(profile,Q))/b} \quad (8)$$

Theorem 1 *The probabilistic model satisfies Property 4.1.*

Proof: We know that the probability $P(L_k = p|Q)$ of point p corresponds to a path ending at p. The theorem can be proven by applying Equation 8. Interested readers should refer to [9] for a detailed proof.

□

In fact, the probability $P(L_k = p|Q)$ of point p corresponds to the best path among the paths ending at p.

Theorem 2 *The probability $P(L_k = p|Q)$ of point p corresponds to the best path among those paths ending at p.*

Proof: The theorem can be proven by induction based on Equation 5. Refer to [9] for a detailed proof.

□

To demonstrate Theorems 1 and 2, suppose we are given the map in Figure 1, and query profile

$$Q = \{(-11.1, 1), (-81.7, \sqrt{2})\}$$

and $\delta_s = 10, \delta_l = 0.5, b_s = 100, b_l = 5$. According to Equation 5, the probability $P(L_2 = (2, 2)|Q) = 0.0824$. And other variables calculated from Equation 5 is $P_0 = 0.04$, $\alpha_1 = 2491.7$ and $\alpha_2 = 3353.7$. Suppose we have two paths ending at point (2,2):

$$path_u = \{(1, 4, 6.7), (1, 3, 18.3), (2, 2, 135.3)\}$$
$$path_v = \{(1, 1, 0.3), (1, 2, 6.7), (2, 2, 135.3)\}$$

Comparing the paths with the query profile, we can have $D_s(path_u, Q) = 1.5$, $D_l(path_u, Q) = 0$ and $D_s(path_v, Q) = 51.6$, $D_l(path_v, Q) = 0.414$. According to Equation 4, $path_u$ is better than $path_v$. Also, from the map we can know that $path_u$ is the best matching path ending at point (2,2).

If $P(L_2 = (2,2)|Q)$ corresponds to $path_v$, we have

$$P(L_2 = (2,2)|Q) = P_0 \cdot \alpha_0 \cdot \alpha_1 \cdot (\frac{1}{200})^2 \cdot (\frac{1}{10})^2 \cdot e^{-\frac{51.6}{100}} \cdot e^{-\frac{0.414}{5}} = 0.0459$$

While if probability $P(L_2 = (2,2)|Q)$ corresponds to $path_u$, we have

$$P(L_2 = (2,2)|Q) = P_0 \cdot \alpha_0 \cdot \alpha_1 \cdot (\frac{1}{200})^2 \cdot (\frac{1}{10})^2 \cdot e^{-\frac{1.5}{100}} \cdot e^{-\frac{0}{5}} = 0.0824$$

As we can see, $path_u$ has a higher probability value than $path_v$. This matches the fact that $path_u$ is better than $path_v$ as illustrated by the distance measurements. And the probability $P(L_2 = (2,2)|Q)$ corresponds to $path_u$ since the value calculated from Equation 5 equals the value calculated by $path_u$. This also matches the fact that $path_u$ is the best matching path ending at point (2,2).

Also, after the two recursions, point (2,2) has probability 0.0824 and point (1,2) has probability 0.0352. We know point (2,2) corresponds to path

$$path_u = \{(1, 4, 6.7), (1, 3, 18.3), (2, 2, 135.3)\}$$

And point (1,2) corresponds to path

$$path_v = \{(1, 1, 0.3), (2, 1, 6.7), (1, 2, 6.7)\}$$

Compared with the query profile, $D_s(path_u, Q) = 1.5$, $D_l(path_u, Q) = 0$ and $D_s(path_v, Q) = 88.2$, $D_l(path_v, Q) = 0$. Obviously $path_u$ is a better path than $path_v$.

We have shown that our probabilistic model satisfies Property 4.1 and that the probability of each point corresponds to its appearance on the best path. In order to allow user-specified error tolerances, we convert the error thresholds δ_s and δ_l into a single probability threshold.

In the worst case, a path has exactly the distance measurements equal to the error thresholds and it starts at a point having the lowest initial probability value, P_{0min}.

$$D_s(profile, Q) = \delta_s, D_l(profile, Q) = \delta_l$$

Assume that there is a point p in the map which corresponds to this worst-case path, then its probability value should be

$$\begin{aligned} &P(L_k = p|Q) \\ &= P_{0min} \cdot (\prod_{i=1}^{k} \alpha_i)(\frac{1}{2b})^{2k} e^{-(D_s(profile,Q)+D_l(profile,Q))/b} \\ &= P_{0min} \cdot (\prod_{i=1}^{k} \alpha_i)(\frac{1}{2b})^{2k} e^{-(\delta_s+\delta_l)/b} \end{aligned}$$

Therefore, the probability threshold $P_{min}^{(k)}$ should be

$$P_{min}^{(k)} = P_{0min} \cdot (\prod_{i=1}^{k} \alpha_i)(\frac{1}{2b})^{2k} e^{-(\delta_s+\delta_l)/b} \quad (9)$$

where P_{0min} is the minimal value of the initial distribution.

Theorem 3 *Any point that has probability $P(L_k = p|Q)$ lower than $P_{min}^{(k)}$ cannot be an endpoint of any matching path.*

Proof: If a point p has $P(L_k = p|Q)$ lower than $P_{min}^{(k)}$, its corresponding path must have at least one of its distance measurements exceeding the threshold. Since the corresponding path is the best path ending at p, all the other paths ending at p are worse, so at least one of the distance measurements also exceeds the corresponding error threshold and, therefore, cannot be matching paths. Thus, point p is not an endpoint of any matching path. □

On the other hand, according to Equation 9, a point having probability higher than $P_{min}^{(k)}$ also may not be a endpoint of any matching path, because its corresponding path can have $D_s + D_l < \delta_s + \delta_l$ while either $D_s > \delta_s$ or $D_l > \delta_l$.

And similarly, for each profile prefix $Q^{(i)}$, we can also get its probability threshold $P_{min}^{(i)}$

$$P_{min}^{(i)} = P_0 \cdot (\prod_{j=1}^{i} \alpha_j)(\frac{1}{2b})^{2i} e^{-(\delta_s+\delta_l)/b} \quad (10)$$

Theorem 4 *Any point that has probability $P(L_i = p|Q^{(i)})$ lower than $P_{min}^{(i)}$ cannot be an endpoint of any matching path of $Q^{(i)}$, which also means that the point cannot be the $(i+1)^{th}$ point on a matching path of Q.*

Proof: The proof is similar to the one of Theorem 3. □

From Theorems 2 and 3, we can know that if point p has probability larger than $P_{min}^{(k)}$, its corresponding best path may be a matching path. However, Theorem 3 also illustrates that among all the other paths ending at point p, there may also be matching paths as long as some other neighboring points of p can propagate a probability value no less than $P_{min}^{(k)}$ to p. Therefore, for each point p, we define its ancestor point set.

Definition 4.1 *The ancestor point set of p, $A(p)$, is the set of neighboring points of p that can propagate a probability value no less than $P_{min}^{(k)}$ to p.*

$$A(p) = \{p'|\alpha_k \cdot f(p,p') \geq P_{min}^{(k)}\},$$
$$f(p,p') = P(L_k = p|(s_k^q, l_k^q), L_{k-1} = p')P(L_{k-1} = p'|Q^{(k-1)})$$

For point p, each point in $A(p)$ corresponds to a possible matching path ending at p including the best one. It is trivial to show that when point p has probability value lower than $P_{min}^{(k)}$, the set $A(p)$ will be empty.

5 Algorithm

In this section, we explain our query processing algorithm based on the probabilistic model. Later, several optimizations are presented to improve the performance.

Our query algorithm is deterministic even though it employs a probabilistic model. As we discussed in Section 4, the probabilistic model works as a scoring function. There are several reasons for us to use this probabilistic model instead of other score (distance) function models in the deterministic form.

- The probabilistic model is well-established, i.e., all resulting values are within the rage $[0, 1]$ regardless of the data. It scores the point and paths in a natural and easy-to-understand way.

- The probabilistic model is more general than scoring functions and could potentially support arbitrary paths (instead of paths restricted to grid segments as assumed here).

Also our algorithm uses dynamic programming to compute probability propagation between neighbors in our probabilistic model as discussed in Section 4.

5.1 Basic Algorithm

Our algorithm consists of two phases. The first phase identifies endpoints of possible matching paths in the map. This would allow us to constrain the search to regions surrounding these end points. Then the second phase searches for matching paths in reverse starting from these endpoints.

Given a query profile Q of size k, in the first phase, the algorithm employs the probabilistic model to find all points having probability value no less than $P^{(k)}_{min}$. According to Theorem 3, these points may be the endpoints of some matching paths. We use $I^{(0)}$ to denote this set of points — the initial candidate point set. By doing so, the algorithm reduces the search space.

In the second phase, the query profile is reversed and the algorithm starts from the points in set $I^{(0)}$ to search matching paths. The probabilistic model is used again. In each iteration, points having probability no less than $P^{(i)}_{min}$ are recorded in $I^{(i)}$ — the i^{th} candidate point set. According to Theorem 4, $I^{(i)}$ contains all points that may be the $(i+1)^{th}$ point of some matching path. After computing all candidate point sets, $I^{(1)}$ to $I^{(k)}$, the algorithm concatenates the points from each set to form a set of candidate paths, which will be validated and returned. Note that since the query profile is reversed in the second phase, the matching paths also need to be reversed to match the original query profile. In both the first and second phase, the calculation of probabilities employs dynamic programming because of the propagation feature of the model.

In fact, if in the first phase we record the intermediate candidate point sets $I^{(i)}$ besides the last one, we can also get the candidate path set at the end and do not need to run the second phase. However, this only works for small maps. For large maps, the intermediate candidate point sets $I^{(i)}$ in phase 1 can become large, which makes the concatenation intractable. The reason is that in the first phase, since we do not have any idea about the endpoints of the matching paths, we assume that all points have the same initial probability P_0. Therefore, the intermediate candidate point sets in the first phase will contain many false positives (i.e., points on mismatching paths). While in the second phase, with the information of possible endpoints of matching paths, only points in $I^{(0)}$ can get a non-zero initial probability and all other points have $P_0 = 0$. Therefore, the candidate point sets in the second phase tend to be much smaller and contain fewer mismatches. By avoiding materializing the large intermediate candidate point sets in the first phase, the algorithm performs work efficiently on large maps.

```
Phase 1
Input:
  • Elevation map $M$
  • Query profile $Q$ of size $k$
  • error thresholds $\delta_s$, $\delta_l$
Output: Initial candidate set $I^{(0)}$
Method:
  1. for all $p \in M$, $P(L_0 = p|Q^{(0)}) = P_0 = 1/|M|$.
  2. $b_s = \delta_s * 10$, $b_l = \delta_l * 10$.
  3. $P^{(0)}_{min} = P_0 \cdot e^{-(\delta_s/b_s + \delta_l/b_l)}$.
  4. for i=1 to k
  5.    Propagate(i)
  6. $I^{(0)} = \{p | p \in M, P(L_k = p|Q) \geq P^{(k)}_{min}\}$.

Phase 2
Input:
  • Elevation map $M$
  • Reversed query profile $Q'$ of size $k$
  • error thresholds $\delta_s$, $\delta_l$
  • Initial candidate point set $I^{(0)}$
Output: Matching paths
Method:
  1. for $p \in I^{(0)}$, $P_0 = 1/|I^{(0)}|$, otherwise $P_0 = 0$.
  2. $b_s = \delta_s * 10$, $b_l = \delta_l * 10$
  3. $P^{(0)}_{min} = \frac{1}{|I^{(0)}|} \cdot e^{-(\delta_s/b_s + \delta_l/b_l)}$.
  4. for i=1 to k
  5.    Propagate(i)
  6.    $I^{(i)} = \{p | p \in M, P(L_i = p|Q'^{(i)}) \geq P^{(i)}_{min}\}$.
  7. Matching paths=Concatenate().

Propagate(i)
  1.    $\alpha_i = 0$.
  2.    for each point $p \in M$
  3.       Calculate $P(L_i = p|Q^{(i)})$ according to Equation 11
  4.       $\alpha_i = \alpha_i + P(L_i = p|Q^{(i)})$.
  5.    for each point $p \in M$
  6.       $P(L_i = p|Q^{(i)}) = \frac{1}{\alpha_i} P(L_i = p|Q^{(i)})$.
  7.    $P^{(i)}_{min} = P^{(i)}_{min} \cdot \frac{1}{2b_s} \cdot \frac{1}{2b_l} \cdot \frac{1}{\alpha_i}$
```

Figure 2. Basic Algorithm

In the algorithm, we use Equation 11 to calculate the probability recursively. The calculation is implemented via a dynamic programming approach. Equation 11 is modified from Equation 5 by omitting α_i. Because we can only calculate the value of α_i after we get the updated value for each point, we calculate it at the end of each iteration.

$$P(L_i = p|Q^{(i)})$$
$$= max_{p'}\{P(L_i = p|(s^q_i, l^q_i), L_{i-1} = p')P(L_{i-1} = p'|Q^{(i-1)})\} \quad (11)$$

The details of the basic algorithm are shown in Figure 2.

In phase 1, the probability distribution is initialized to be uniform in step 1. $P^{(0)}_{min}$ is set according to the lowest initial probability and the error tolerance in step 3 in order to estimate the worst case. Function $Propagate(i)$ updates the probability distribution and $P^{(i)}_{min}$ recursively according to each profile prefix. Note that Step 3 of $Propagate(i)$ uses dynamic programming. In step 6, the points of $I^{(0)}$ is selected according to their probability value and $P_{min}^{(k)}$.

In phase 2, the main structure is similar with a few small differences. In step 1, the initial probability is set according to $I^{(0)}$, so that only points in $I^{(0)}$ get a non-zero initial probability. In step 6, the candidate point set $I^{(i)}$ for each iteration is recorded. In step 7, the function $Concatenate()$ concatenates all candidate points and returns the matching paths.

In the algorithm, when candidate point sets are generated in step 6 of phase 1 and 2, the ancestor point sets $A(p)$ of all candidates are also recorded. In step 7 of the second phase, $Concatenate()$ appends candidate points according to their ancestor point sets and prunes mismatching paths. The details of $Concatenate()$ are shown in Figure 3.

```
function $Concatenate()$
Input:
  • Candidate point sets $\{I^{(0)}, ..., I^{(k)}\}$
  • Ancestor point set $A(p)$ for each candidate point.
  • $\delta_s$,$\delta_l$.
Output: Matching paths set MPath
Method:
1. MPath $= \emptyset$.
2. Insert all points $p \in I^{(0)}$ into MPath as the initial paths.
3. for $i = 1$ to $k$
4.     for each point p in $I^{(i)}$
5.         for each path in MPath
6.             if the last point of the path, $p_l$, belongs to $A(p)$
7.                 Concatenate point $p$ with the path.
8.     Remove paths in MPath not extended in this turn.
9.     Remove paths in MPath whose $D_s$, $D_l$ exceed $\delta_s$ or $\delta_l$
```

Figure 3. function $Concatenate()$

The complexity of the basic algorithm is $O(|M| \cdot k + R)$, where R is the number of matching paths. Now we will prove the correctness of our algorithm.

Theorem 5 *Given a query profile Q, the error tolerance δ_s and δ_l, all matching paths can be found by the algorithm.*

Proof: For any matching path consisting of points $\{p_0, ..., p_k\}$, since its profile satisfies Equation 1 and 2, we know that

$$D_s(profile, Q) + D_l(profile, Q) \leq \delta_s + \delta_l \qquad (12)$$

1. We prove that p_k is included in set $I^{(0)}$. According to Theorem 2, Equations 8 and 12, we know that

   $$P(L_k = p_k|Q)$$
   $$\geq P_0 \cdot (\prod_{i=1}^{k} \alpha_i)(\frac{1}{2b})^{2k} e^{-(D_s(profile,Q)+D_l(profile,Q))/b}$$
   $$\geq P_0 \cdot (\prod_{i=1}^{k} \alpha_i)(\frac{1}{2b})^{2k} e^{-(\delta_s+\delta_l)/b} = P_{min}^{(k)}$$

   So that point p_k has probability value no less than $P_{min}^{(k)}$ and is included in $I^{(0)}$.

2. We prove that each remaining point on the path is also included in the corresponding candidate point set. Since the query profile is reversed in the second phase of the algorithm, we also reverse path $\{p_0, ..., p_k\}$ for convenience. Let Q' be the reversed query profile. Let $\{p'_0, ..., p'_k\}$ be the reversed path, $p'_i = p_{k-i}$, and $profile'$ be its reversed profile. We have already proved that p'_0 is included in $I^{(0)}$. By the same token, for each point p'_i, we have

   $$P(L_i = p'_i|Q'^{(i)})$$
   $$\geq P_0 \cdot (\prod_{j=1}^{i} \alpha_j)(\frac{1}{2b})^{2i} e^{-(\sum_{j=1}^{i}|s'_j - s'^q_j| + \sum_{j=1}^{i}|l'_j - l'^q_j|)/b}$$
   $$\geq P_0 \cdot (\prod_{j=1}^{i} \alpha_j)(\frac{1}{2b})^{2i} e^{-(D_s(profile',Q')+D_l(profile',Q'))/b}$$
   $$\geq P_0 \cdot (\prod_{j=1}^{i} \alpha_j)(\frac{1}{2b})^{2i} e^{-(\delta_s+\delta_l)/b} = P_{min}^{(i)}$$

   So each point p'_i has probability value no less than $P_{min}^{(i)}$ and is included in $I^{(i)}$.

3. We prove that each point p'_i is included in the ancestor point set of p'_{i+1}.

$$\alpha_{i+1} \cdot P(L_{i+1} = p'_{i+1}|(s'^q_{i+1}, l'^q_{i+1}), L_i = p'_i)P(L_i = p'_i|Q'^{(i)})$$
$$\geq \alpha_{i+1} \cdot (\frac{1}{2b})^2 \cdot e^{-(|s'_{i+1} - s'^q_{i+1}| + |l'_{i+1} - l'^q_{i+1}|)/b}$$
$$\cdot P_0 \cdot (\prod_{j=1}^{i} \alpha_j)(\frac{1}{2b})^{2i} e^{-(\sum_{j=1}^{i}|s'_j - s'^q_j| + \sum_{j=1}^{i}|l'_j - l'^q_j|)/b}$$
$$= P_0 \cdot (\prod_{j=1}^{i+1} \alpha_j)(\frac{1}{2b})^{2(i+1)} e^{-(\sum_{j=1}^{i+1}|s'_j - s'^q_j| + \sum_{j=1}^{i+1}|l'_j - l'^q_j|)/b}$$
$$\geq P_0 \cdot (\prod_{j=1}^{i+1} \alpha_j)(\frac{1}{2b})^{2(i+1)} e^{-(D_s(profile',Q')+D_l(profile',Q'))/b}$$
$$\geq P_0 \cdot (\prod_{j=1}^{i+1} \alpha_j)(\frac{1}{2b})^{2(i+1)} e^{-(\delta_s+\delta_l)/b} = P_{min}^{(i+1)}$$

   According to Definition 4.1, for each point p'_i, $0 \leq i < k$, we have $p'_i \in A(p'_{i+1})$.

Therefore, path $\{p_0, ..., p_k\}$ must be able to be concatenated by the $Concatenate()$ function and be returned as a matching path.

□

5.2 Optimizations

In this section, we discuss several optimizations to the basic algorithm.

5.2.1 Selective Calculation

In the basic algorithm, a major overhead is to calculate the probability values for each point. For large elevation maps, it takes a long time to do so for all points.

According to the propagation relationship between candidate points in the algorithm, a point in candidate point set $I^{(i)}$ must be the neighboring point of a point in $I^{(i-1)}$. Therefore, if we know that point p is a candidate point of $I^{(i)}$, we only need to focus on the small area surrounding p to find candidate points p' in subsequent steps.

To expedite the computation, we can partition the map into smaller regions and only for points in regions containing candidate points, the algorithm will calculate their probability value recursively. For example, in our experiments, we partition a 2000 $\times$ 2000 map into a list of 50 $\times$ 50 regions. In order to handle candidate points near boundary of regions, we enlarge each region slightly according to the size of query profile, thus including a small overlap.

The efficiency of this *selective calculation* depends on the distribution and number of candidate points. If most small regions contain candidate points, there will be little improvement over the basic algorithm. Therefore, before using selective calculation, we check the number of candidate points. If the number of candidate points is small, there is a high chance that the number of regions containing candidate points is also small and selective calculation would be likely to be effective.

Selective calculation can be applied to both phases in different ways. In phase 1, a check step will be inserted after step 5. If the number of points having probability no less than $P_{min}^{(i)}$ is small, the algorithm will turn to use selective calculation. Note that the algorithm only needs to know the number of points, the actual candidate sets are not materialized. When the profile is long, the last several candidate point sets will usually contain few points. In this case, selective calculation can show its efficiency in phase 1.

And in phase 2, a check step is inserted after step 3. If the number of initial candidate points is small, the algorithm turns to use selective calculation. When the user-specified error tolerance is tight, the initial candidate set would contain few points. In this case, it is efficient to use selective calculation. We show the performance of selective calculation in the experiment section.

5.2.2 Reversed Concatenation

In $Concatenate()$, candidate points are concatenated from $I^{(0)}$ to $I^{(k)}$. However, we observe that many candidate points in early candidate set can not be extended to form a complete matching path when the concatenation continues. The reason is that an intermediate path ending at an early candidate point may have already reached the maximal error tolerance to the query profile and any further step will make it a mismatching path. While points in later candidate sets have a higher chance to form a complete matching path. Also the later candidate sets are usually of smaller size.

Therefore, instead of starting concatenation from $I^{(0)}$, we reverse the concatenation and start from $I^{(k)}$, the last candidate set. By doing so, the number of candidate paths tested in the intermediate steps decreases dramatically. We compare the performance of reversed concatenation with the normal concatenation in the experiments.

5.2.3 Pre-processing

In the algorithm, the slopes and distances of segments between points and their neighboring points are frequently used. It is inefficient to calculate these values each time when a new query comes. Therefore, for each map, we conduct a pre-processing to calculate the slopes and distances around each point and store them in matrix. Then in the algorithm, these values can be quickly loaded from the matrix. The computation time can be reduced to about 60% by pre-processing according to experiment performance.

6 Experiments

In this section, we compare our algorithm with an alternative method based on a traditional indexing structure. We demonstrate the efficiency and scalability of our algorithm and the effectiveness of the proposed optimization methods.

Dataset

We used a 2000 $\times$ 2000 elevation map downloaded from the North Carolina Floodplain Mapping Program [1]. The xy view of the map is shown in Figure 4(a). We also generate

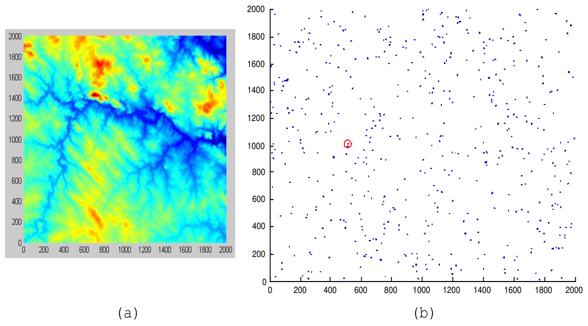


Figure 4. xy view of the elevation map (a) and the matching paths (b) of the profile in Figure 5 Left

several smaller maps that are regions of the 2000 $\times$ 2000 map.

An example query profile of size $k = 7$ is shown in Figure 5. We set $\delta_s = 0.5$, $\delta_l = 0.5$. A total of 763 matching paths are returned, and their spatial distribution is plotted in Figure 4(b). The path in the circle is exactly the path used to generate the query profile. The shapes of the the matching paths are shown in Figure 5. As we can see, the matching paths exhibit profiles similar to the query profile.

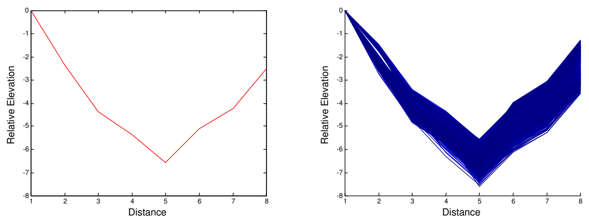


Figure 5. Left: shape of the query profile. Right: shape of the matching paths

Alternative Method

We implemented an alternative method based on the traditional index structure B+ tree.

Each segment in the map (including horizontal, vertical and diagonal segments) is indexed by a B+tree with its slope value as the index key. The segment length is not used as the key since it is either 1 (for horizontal and vertical segments) or $\sqrt{2}$ (for diagonal segments) in an grid format elevation map. A profile query with error tolerance δ_s is decomposed into a set of k segment queries using their slope values, where k is the profile size. δ_s/k is used as the error tolerance in each segment query. The returned matching segments will then be assembled into matching paths. Note that

[1] $http://www.ncfloodmaps.com/default\_swf.asp$

the alternative method can only find a subset of all matching paths. It would entail much longer computation in order to find all matching paths and are intractable for even short profile and moderate error tolerance. Therefore, we only compared the performance of the described algorithm with our approach. In the following discussion, we refer to this alternative method as the $B^+segment$ method.

Other spatial index structures like R-tree can store each possible paths in the map as a high dimensional point and perform the query. However, this is only feasible when both the map and the size of query profile are very small. With a 2000×2000 map and a profile of size 7, the number of possible paths in the map can be up to $O(10^{12})$.

Parameters

We evaluated the performance of our algorithm with respect to four parameters: map size m, query profile size k and error tolerance δ_s, δ_l. The prototype was implemented in MATLAB and all experiments were conducted on a PC with P4 3GHz and 1G main memory. In Section 6.1, we compare our method with the $B^+segment$ method on a smaller map because, with reasonable running time, $B^+segment$ can only answer queries on small maps with limited error tolerance. In Section 6.2, we test the performance of our algorithm with various parameters. In Section 6.3, we show the efficiency of the optimization methods.

6.1 Comparison with alternative method

As we mentioned in the previous section, $B^+segment$ can only handle error tolerance segment by segment, so we evenly distribute δ_s among k segments. It is obvious that the set of matching paths found by $B^+segment$ is a subset of the matching paths. We use a map of size 500×500, since $B^+segment$ is unable to handle large maps. We varied the value of δ_s to compare the runtime performance and number of paths found. We set $\delta_l = 0$ and $k = 7$.

As shown in Figure 6, our approach is orders of magnitude faster except when $\delta_s = 0$. The number of paths found by the two methods is also provided in the figure. As we expected, $B^+segment$ is not able to find all matching paths.

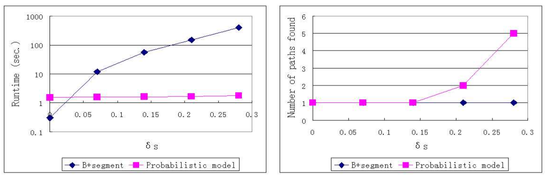


Figure 6. Runtime of our algorithm remains almost constant while runtime of $B^+segment$ increases exponentially when increasing δ_s, and k=7, δ_l=0.5,m=2.5 $\cdot$ 10^5. And $B^+segment$ cannot find all the matching paths.

The reason for the poor performance of $B^+segment$ is that data structures like B+tree only contain information about the segment's length and slope, while the connection (adjacency) information of the segments is not included. Lots of mismatching segments can only be pruned during concatenation, after they are returned by B+tree. This makes the method very inefficient on large maps and with high error tolerance.

6.2 Algorithm Performance

In this section, we vary the values of four parameters to test our algorithm. Table 1 lists the parameters.

Table 1. Parameter range and default value

parameter	range	default value
k	$\{7, 11, 15, 19, 23\}$	7
δ_s	$\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$	0.5
δ_l	$\{0, 0.5\}$	0.5
m	$\{2.5 \cdot 10^5, 5 \cdot 10^5, 1 \cdot 10^6\}$ $\{2 \cdot 10^6, 4 \cdot 10^6\}$	$4 \cdot 10^6$

We test two kinds of query profiles: profile generated from an actual path in the map and profile randomly generated. Unless otherwise noted, we use a profile from the map as the query profile.

6.2.1 Varying δ_s and δ_l

The values of m and k are set to their default values in these experiments. Since a segment in the map is either of length 1 or $\sqrt{2}$, we only set two values for δ_l: 0 and 0.5 and vary δ_s from 0.1 to 0.6.

The runtime and number of matching paths for different δ_s and δ_l settings are shown in Figure 7.

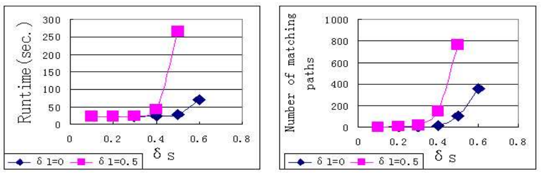


Figure 7. Runtime and number of matching paths of sampled profiles increase exponentially when increasing δ_s, δ_l=$\{0, 0.5\}$, m=4 $\cdot$ 10^6, k=7.

As we can see in Figure 7, with larger δ_s and δ_l, more paths may match the query profile and the query takes longer time to run. In fact, the runtime is linear to the number of paths returned as shown in Figure 8.

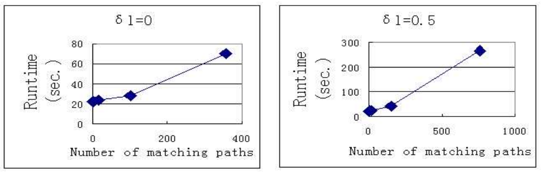


Figure 8. Runtime of our algorithm increases linearly with the increase of number of matching paths of sampled profiles by increasing δ_s, $m = 4 \cdot 10^6$, $k = 7$

6.2.2 Varying m

We applied the probabilistic model to maps of different size. Parameters k, δ_s and δ_l are set to their default values.

The runtime and number of matching paths on maps of increasing sizes are shown in Figure 9. Both the runtime and number of matching paths are linear to the map size.

6.2.3 Varying k

We next varied the query profile size and tested the performance. We selected a path consisting of 24 points in the map and use its profile as the query profile. The profiles of

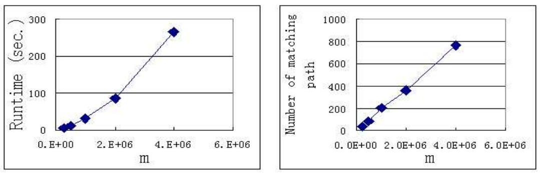


Figure 9. Runtime and number of matching paths of sampled profiles increase linearly when increasing map size m, δ_s=δ_l=0.5, k=7.

smaller size are its corresponding profile prefixes. Parameters m, δ_s and δ_l are set to their default values.

The runtime and number of matching paths for each query profile are shown in Figure 10. Except when $k = 7$, the runtime is linear to the size of query profile. The reason that the algorithm takes much more time when $k = 7$ is that there are many more matching paths, which takes longer time to process. According to the complexity of our algorithm, when the number of matching paths is relatively small, the map size and profile size will play dominant roles in the runtime. As we can see in Figure 10, when $k = 11, 15, 19, 23$, the number of matching paths is less than 10, and the corresponding runtime is linear to profile size k.

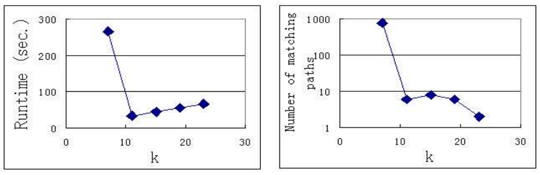


Figure 10. Runtime increases linearly when increasing profile size k, and δ_s=δ_l=0.5, m=$4 \cdot 10^6$, except when there are large number of matching paths. Number of matching paths decreases dramatically when increasing k.

Instead of using the profile generated from a path in the map, we also tested it with random query profile, and varied the parameter δ_s to evaluate the algorithm's performance. All other parameters were set to their default values.

The runtime and number of matching paths are shown in Figure 11. As we increased δ_s, the number of matching paths increased exponentially and so does the runtime. Compared to samples queries, under the same parameter setting, random profiles have similar runtime performance. As shown in Figure 12, the runtime is also linear to the number of paths returned.

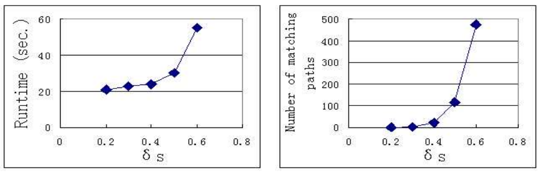


Figure 11. Runtime and number of matching paths of random profiles increase exponentially when increasing δ_s, δ_l=0.5, m=$4 \cdot 10^6$, k=7

6.3 Efficiency of Optimizations

In this section, we show the efficiency of the optimization methods. Each of the optimization methods will be added to the basic algorithm separately and their performance will be compared with the basic algorithm.

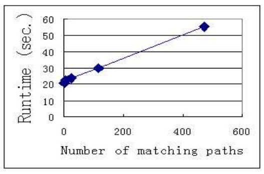
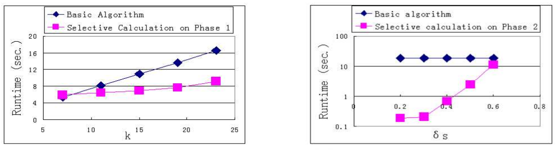


Figure 12. Runtime of our algorithm increases linearly with the increase of number of matching paths of random profiles by increasing δ_s, $\delta_l = 0.5$, $m = 4 \cdot 10^6$, k=7

As discussed in Section 5, adding selective calculation in phase 1 is most efficient when the query profile is large. Therefore we varied the value of k and compared the runtime of phase 1 only. For the other parameters, $\delta_s = 0.5$, $\delta_l = 0$, and $m = 10^6$. As we can see in Figure 13(a), the optimization can save about 50% runtime of phase 1 when the query profile has $k = 23$. When the profile is of very small size, the optimization does not improve the runtime performance substantially.



(a) Comparison of runtime on Phase 1 when varying k. Selective calculation save up to 50% time on phase 1. $\delta_s = 0.5$, $\delta_l = 0$, $m = 10^6$

(b) Comparison of runtime on Phase 2 when varying δ_s. With Selective calculation, phase 2 can be orders of magnitudes faster. $k = 7$, $\delta_l = 0$, $m = 10^6$

Figure 13. Comparing the runtime of basic algorithm and selective calculation

Adding selective calculation to phase 2 is efficient when the error tolerance is relatively small. Therefore, we varied δ_s to show the efficiency of selective calculation on phase 2. We set δ_l=0, $k = 7$, and $m = 10^6$. We only compared the runtime of phase 2 as shown in Figure 13(b). The basic algorithm always takes the same amount of time in phase 2 no matter what δ_s is given. With this optimization, phase 2 is spedup by orders of magnitude, especially when δ_s is small.

Next, we added the reversed concatenation function to the basic algorithm in place of normal concatenation. In order to show the efficiency of reversed concatenation, we compare the number of paths generated in each intermediate iteration by normal and reversed concatenation. The parameters are set as follows, $\delta_s = \delta_l = 0.5$, $k = 7$ and $m = 2.5 \cdot 10^5$ and the query profile is random. The comparison of number of paths is shown in Figure 14. As we can see, the number of paths generated is dramatically reduced especially in the early iterations.

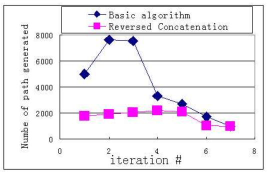


Figure 14. Number of paths generated in intermediate iterations is dramatically reduced by using reserved concatenation compared to normal concatenation, $k = 7$, $\delta_s = \delta_l = 0.5$, $m = 2.5 \cdot 10^5$

From the experiments shown in this section, we can see that our probabilistic model and algorithm can handle profile query efficiently and are much better than alternative methods based on traditional indexing structures.

7 Application of Profile Query

In this section, we use the algorithm to solve a Map Registration problem. Suppose we have two raster maps, one is of size 1000×1000 and the other is of size 20×20 as shown in Figure 15(a) and (b). The small map is a sub-region of the big one, therefore we want to find the location of the small map in the big map (the locations of its left-bottom and right-up corners).

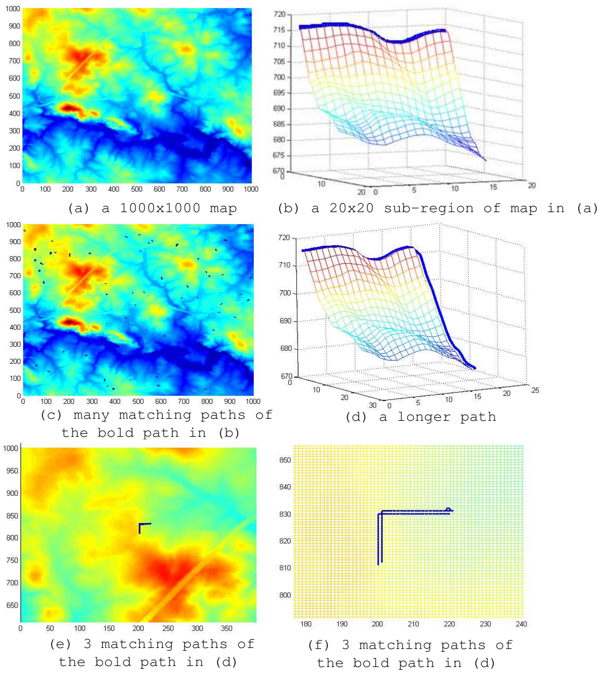

(a) a 1000x1000 map (b) a 20x20 sub-region of map in (a)
(c) many matching paths of the bold path in (b) (d) a longer path
(e) 3 matching paths of the bold path in (d) (f) 3 matching paths of the bold path in (d)

Figure 15. Example of Map Registration

To solve this map registration problem by profile query algorithm, we select a path in the small map, generate its profile and then search the profile in the big map. If the selected path is long enough, its profile will probably be unique and the query algorithm can return the only corresponding path which helps us to locate the sub-region.

A path consisting of 20 points, which is shown as the bold curve in Figure 15(b), was selected. The query results are shown in Figure 15(c). Since the selected path is not long enough, several paths in the big map have the similar profile so that we can not locate the small map. Therefore, we select a longer path which consists of 40 points as shown in Figure 15(d). This time, the query algorithm only returns three paths in Figure 15(e) and (f). According to the returned paths, the small maps is located at points $(811, 201)$ and $(830, 220)$ or points $(812, 202)$ and $(831, 221)$ (the locations of its left-bottom and right-up corners). In fact, the small map in 15(b) is located at points $(811.5, 201.5)$ and $(830.5, 220.5)$. Since the query algorithm only returns paths consisting of grid segments, it locates the sub-region at the closest grid points.

We tested the algorithm with more sub-regions selected randomly from the big map. For most of the sub-regions, a path consisting of 40 points is enough to uniquely locate them in the big map no matter the sizes of the sub-regions.

A brute-force search of the sub-region by comparing each possible path (considering x-y shape) with the selected path from the sub-region can have similar runtime performance with our basic algorithm. However, with the optimization methods in Section 5.2, our query algorithm can be orders of magnitude faster.

8 Conclusion

In this paper, we introduce the problem of profile query on elevation map. Given a query profile, the objective is to find matching paths in the map within some error tolerance. We developed a probabilistic model which has been demonstrated to be very efficient via real experiments. Comparing to alternative method based on traditional index structure, our algorithm is orders of magnitude faster. Future work includes supporting query profile expressed in more general format (than a list of segments of standard sizes), applying the probabilistic model to other types of terrain maps like Triangulated Irregular Network (TIN), and handling multi-resolution maps in a hierarchical structure to further speedup performance on huge maps.

Acknowledgement

This research was supported through NGA/DARPA grant HM1582-05-2-0003.

References

[1] J. A. Castellanos, J. D. Tardss, and J. D. Tardos. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. Kluwer Academic Publishers, Boston, 2000.

[2] C. du Mouza and P. Rigaux. Mobility patterns. In *STDBM '04 Conference Proceedings*, pages 1–8, 2004.

[3] D. Fox, W. Burgard, and S. Thrum. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*.

[4] R. H. Guting, M. H. Bohlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.*, 25(1):1–42, 2000.

[5] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84 Conference Proceedings*, pages 47–57, 1984.

[6] M. Hadjieleftheriou, G. Kollios, P. Bakalov, and V. J. Tsotras. Complex spatio-temporal pattern queries. In *31st VLDB Conference Proceedings*, 2005.

[7] M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos. Efficient indexing of spatiotemporal objects. In *EDBT '02 Conference Proceedings*, pages 251–268, 2002.

[8] I. Nourbakhsh, R. Powers, and S. Birchfield. Dervish an office-navigating robot. *AI Magazine*.

[9] F. Pan, W. Wang, and L. McMillan. Accelerating profile queries in elevation maps. *Technical Report: TR06-018*, 2006.

[10] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In *VLDB '00 Conference Proceedings*, pages 395–406, 2000.

[11] Y. Tao and D. Papadias. Mv3r-tree: A spatio-temporal access method for timestamp and interval queries. In *The VLDB Journal*, pages 431–440, 2001.