Proc. the 17th Japan Conference on Discrete and Computatoinal Geometry and Graphs (JCDCGG 2014), pp.5-6, 2-14 (Sep. 15 - 16, 2014, Tokyo Science University, Kagurazaka, Tokyo)

Bit-parallel Approximate Trajectory Matching for 2-dimensional Trajectory Data

Hirohito Sasakawa* **Masahiro Yamamoto*** **Kazuhiro Kurita†** **Hiroki Arimura***

Abstract: We consider the 2-d approximate trajectory matching problem, and show that it can be solved in $O(n(\frac{m \log\log n}{\log n}))$ time using linear space.

1. Introduction

By rapid progress of sensing and communication technologies such as GPS, massive amount and species of trajectory data in 2-d and higher dimension have been available, and efficient methods for processing them have attracted much attention. In this paper, we introduce the 2-d approximate trajectory matching problem w.r.t. L_∞ distance. It is the trajectory counterpart of the k-difference approximate string matching problem [5] in which edit distance is replaced with the maximum of L_∞ distance between points. Then, we present an efficient algorithm Shift-AndTrajMatch for the problem. The algorithm finds all occurrences of a pattern trajectory of length m in an input text trajectory of length n in $O(n(\frac{m \log\log n}{\log n}))$ time using linear space by combining bit-parallelism [2, 6] and block decomposition technique for discrete Fréchet distance matching [1].

2. Preliminaries

In this section, we give basic definitions and notations on our trajectory matching problem. For the definition not found here, see textbooks [3, 5].

2.1 Basic definition

We give the definition of trajectory data, according to [4]. Let $\mathbb{R}^2$ be a **2-dimensional plane** and $p = (x, y) \in \mathbb{R}^2$ be a **point** on $\mathbb{R}^2$. We define L_∞ **distance** between points p and q by

$$d(p,\ q) = \max\{|p.x - q.x|,\ |p.y - q.y|\}.$$

Let $S = s_1 \dots s_n$ be a **trajectory** on $\mathbb{R}^2$ where $S[i] = s_i \in \mathbb{R}^2$ for every $1 \le i \le n$.

2.2 Approximate trajectory matching

Let r be a non-negative real number. A **text trajectory** (or text) $T = t_1 \dots t_n$ of length n and a **pattern trajectory** (or pattern) $P = p_1 \dots p_m$ of length m are trajectory on $\mathbb{R}^2$. We say that a pattern P **occurs** in a text T if there exits some $1 \le k \le n$ such that $d(p_i, t_{k+i-1}) \le r$ for every $1 \le i \le m$. Then, the index k is called the **occurrence position** of P in T. Now, we define the 2-d approximate trajectory matching problem below:

* Grad. School of IST, Hokkaido University
† Faculty of Engineering, Hokkaido University
Contact: sasakawa@ist.hokudai.ac.jp

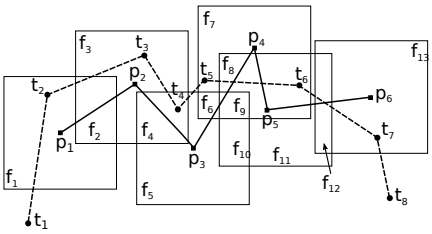

Figure 1 An example of occurrence of the trajectory matching problem and an arrangement $\mathcal{A}(P)$

Definition 1 *The **2-d approximate trajectory matching problem** is, given a pattern P of length m and a text T of length n, to find all occurrence positions of P in T.*

Figure 1 shows an example of an occurrence the problem. In this figure, $P = p_1 \dots p_6$ occurs $t_2 \dots t_7$.

2.3 Model of computation

As a computation model, we adopt unit-cost word RAM with $w = \Omega(\log n)$ bit words. We assume that standard bitwise and arithmetic operations in a word are executed in $O(1)$ time.

3. Efficient trajectory matching algorithm

In this section, we give an efficient algorithm for our trajectory matching problem. The proposed algorithm Shift-AndTrajMatch is an extension of a bit-parallel string matching algorithm called Shift-And [2,6] on 2-dimensional trajectory matching.

In Algorithm 1, we present the proposed trajectory matching algorithm. This algorithm constructs nondeterministic finite automaton (NFA for short) accepts an input pattern P, and then simulates transitions of the NFA by reading a point in text trajectory T one by one. In preprocessing, the algorithm constructs a set of **bitmasks** representing an NFA of P and stores them in **point location index** to extract a requested bitmask effectively. In searching phase, it perform bit-parallel NFA simulation using the bitmasks. The details of each phase is shown below.

Preprocessing: We consider a set of squares $\mathcal{D}(P) = \{D_1, \dots, D_m\}$ of radius r centered by each point in an input pattern trajectory $P = p_1 \dots p_m$. For every $i = 1, \dots, m$, $D_i \in \mathcal{D}(P)$ is an square centered by p_i. Next, we consider an **arrangement** $\mathcal{A}(P)$ of $\mathcal{D}(P)$ (Figure 1). In an arrangement, a **vertex** is a intersection of segments, a **edge** a segment between two vertexes, and a **face** is a region enclosed by edges. Then,

Algorithm 1 Shift-AndTrajMatch(P, T)

```
   Preprocessing
 1: for i ∈ 0 ... ⌈m/h⌉ − 1 do
 2:     B[i] ← p_{hi} ... p_{h(i+1)−1}
 3:     PL[i] ← ConstructIndex(B[i])
   Searching
 4: S ← 0^m
 5: for pos ∈ 1...n do
 6:     for j ∈ 0 ... ⌈m/h⌉ − 1 do
 7:         M[j] ← PL[j].PointLocation(t_pos)
 8:     S ← ((S ≪ 1) | 0^{m−1}1) & M
 9:     if S & 10^{m−1} ≠ 0^m then
10:         report an occurrence at pos − m + 1
```

we have the following lemma on arrangement.

Lemma 1 *Let P be a pattern trajectory of length m and $\mathcal{A}(P)$ be an arrangement determined by P. The number of faces in $\mathcal{A}(P)$ are $O(m^2)$.*

$\mathcal{F}(P)$ denotes a set of faces determined by the arrangement $\mathcal{A}(P)$, and $\mathcal{F}(P) = \{f_1, \ldots, f_K\}$ $(K = O(m^2))$.

Next, we consider **point location index** $\mathcal{P}L$ supporting following operations: **Index construction** $ConstructIndex(P)$: Storing all the squares in the arrangement $\mathcal{A}(P)$ into the structure: **Point location** $PointLocation(v)$: Return bitmask representation of a set of squares that includes the face $f_k \in \mathcal{F}(P)$ which holds the input point v. In this algorithm, we use a point location index structure satisfying following lemma:

Lemma 2 *Let $w = \Omega(\log n)$ be the length of a computer word. For any arrangement of m squares, there exists an index structure that perform an point location operation in $O(\log m + \lceil m/w \rceil)$ time using $O(m \log m + \lceil m^2/w \rceil)$ preprocessing time and $O(\lceil m^2/w \rceil)$ word.*

In preprocessing, we divide an input pattern into blocks whose size is $h = \log n$ points and then it constructs point location index independently for each block.

Searching: First, the algorithm reads a point in a input text trajectory one by one, and get a set of bitmasks corresponding to the point by querying point location indexes of each block. Next, it simulates transitions of NFA by bit-parallelism on line 8 using the bitmasks.

4. Complexity analysis

In this section, we analyze the time and space complexity of the proposed algorithm.

First of all, we consider space complexity. Shift-AndTrajMatch has point location indexes for each block. The number of blocks is $\lceil m/h \rceil$ and a point location index that contains h squares consumes $O((h)^2/w) = O(\log n)$ words. Therefore, the total space of these indexes is $\lceil m/h \rceil \times O(\log n) = O(m)$ words.

Next, we consider time complexities for preprocessing and searching. For preprocessing, construction of $\lceil m/h \rceil$ point location indexes consumes $O(\lceil m/h \rceil (h \log h + \lceil h^2/w \rceil)) = O(m \log \log n)$ time. For searching, a single transition invokes $\lceil m/h \rceil$ point location operation on line 7 and bit-parallel operation on line 8. A point location requires $O(\log h)$ time, and bit-parallel operation needs $O(\lceil m/w \rceil)$. Therefore, Shift-AndTrajMatch requires $O(n \lceil m/h \rceil (\log h + \lceil m/w \rceil)) = O(nm \log \log n / \log n)$ to process whole text T of length n. From the discussion above, we show the main result of this paper.

Theorem 1 *Given a pattern trajectory P of length m, a text trajectory T of length n, and non-negative number r, the algorithm Shift-AndTrajMatch solves the 2-d approximate trajectory matching problem in $O(\frac{nm \log \log n}{\log n})$ time using $O(m)$ space and $O(m \log \log n)$ preprocessing time.*

5. Conclusion

In this paper, we consider the 2-d approximate trajectory matching problem. The proposed algorithm Shift-AndTrajMatch solves the problem efficiently by combining a point location index and bit-parallel string matching technique. As the future work, implementing our algorithm and performing computational experiments on the real world GPS data are important task.

References

[1] Pankaj K Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43(2):429–449, 2014.

[2] Ricardo Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. *Commun. ACM*, 35(10):74–82, October 1992.

[3] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[4] Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

[5] Gonzalo Navarro and Mathieu Raffinot. *Flexible Pattern Matching in Strings: Practical On-line Search Algorithms for Texts and Biological Sequences*. Cambridge University Press, New York, NY, USA, 2002.

[6] Sun Wu and Udi Manber. Fast text searching: Allowing errors. *Commun. ACM*, 35(10):83–91, October 1992.

JCDCG² 2014 The 17th Japan Conference on Discrete and Computational Geometry and Graphs

Sep. 15-16, 2014

Tokyo University of Science, Kagurazaka Campus

The 17th Japan Conference on Discrete and Computational Geometry and Graphs

Sponsors

Tokyo University of Science
Tokai University

Contents

On Geometric Cycles of Point Sets with Many Intersections 1
*Jorge Cravioto, José Luis, Álvarez Rebollar, and Jorge Urrutia**

Long Monotonic Paths in Weighted Point Sets 3
Toshinori Sakai

Bit-parallel Approximate Trajectory Matching for 2-dimensional Trajectory Data 5
Hirohito Sasakawa, Masahiro Yamamoto, Kazuhiro Kurita, and Hiroki Arimura*

Polynomial-Time Algorithm for Sliding Tokens on Trees 7
Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada*

Continuously Flattening Polyhedra with Two Rigid Adjacent Faces —Part II— 9
*Jin-ichi Itoh and Chie Nara**

Comparison between Curved-Surface Tessellations and the Voronoi Diagrams Using Planar Photographic Images 11
Supanut Chaidee and Kokichi Sugihara*

Computing a Dominating Set on Grids 13
Pochara Patthamalai and Xuehou Tan*

OR-Game of Nim-type Games 15
Hiro Ito, Mikio Kano, and Keita Sekimoto*

Invited Talk: Variations on the tree graph for abstract graphs and for geometric graphs 17
Eduardo Rivera-Campo

Optimally Bracing Frameworks of Union of Space-filling Convex Polyhedra 19
Yoshihiko Ito, Yuki Kobayashi, Yuya Higashikawa, Naoki Katoh, Takashi Horiyama, Jin-ichi Itoh, and Chie Nara*

Optimally Bracing Grid Frameworks with Holes 21
Yoshihiko Ito, Yuki Kobayashi, Yuya Higashikawa, Naoki Katoh, Sheung-Hung Poon, and Maria Saumell*

Complete Analysis for Dudeney's Haberdasher's Problem 23
*Jin Akiyama and Hyunwoo Seong**

Common Unfolding of Three Different Boxes of Surface Area 30 25
Xu Dawei, Toshihiro Shirakawa, and Ryuhei Uehara*

Degree Sequentially Unique Graphs 27
Narong Punnim

Generalized Jankens Including Ties 29
Hiro Ito and Yoshinao Shiono*