Approximate pattern matching under rigid motion

Hein Kruger

28 March 2006
Introduction

- Problem:
  given two sets $A$ and $B$ of points, line segments or (filled-in) polygons in $\mathbb{R}^2$, is there a subset of $B$ that is an approximate copy of $A$, perhaps under some Euclidean transformation?

- Solution:
  find the smallest $\epsilon > 0$ such that some rigid motion brings every element of $A$ to within a distance $\epsilon$ of $B$.

- one-way minimum Hausdorff distance problem:
  find the smallest $\epsilon > 0$ such that there is some Euclidean transformation $g$ with $d_H(g(A), B) \leq \epsilon$ [Chew et al. 1997].

- $O(m^3n^2 \log^2 mn)$ for point patterns.

- $O(m^3n^3 \log^2 mn)$ for line segments.
Approach

- Solve decision problem for a given $\epsilon > 0$, is there a Euclidean transformation $g$ such that $d_H(g(A), B) \leq \epsilon$?

- Use decision problem in binary search: similar to algorithm to approximate Frechet distance.

- Will only cover decision problem
Definitions and Notation

Let \( \{a_1, \ldots, a_m\} \) and \( \{b_1, \ldots, b_n\} \), be finite sets of points, line segments or (filled-in) triangles in \( \mathbb{R}^2 \). Now let \( A = \bigcup_{i=1}^{m} a_i \) and \( B = \bigcup_{i=1}^{n} b_i \).

- one-way Hausdorff distance decision problem: Given \( A \) and \( B \), and \( \epsilon > 0 \), is there a transformation \( g \in E_2 \) (where \( E_2 \) represents the set of all Euclidean transformations) such that \( \overrightarrow{d_H}(g(A), B) \leq \epsilon? \)
- \( B^\epsilon \) is the \( \epsilon \)-neighbourhood of \( B \): All points that lie within distance \( \epsilon \) of \( B \).
- \( B_i^\epsilon \) denotes the set of all translations that take \( a_i \) to within \( B^\epsilon \).
- We parameterise \( B_i^\epsilon \) by \( \theta \) (angle of rotation): \( B_i^\epsilon(\theta) \) is the set of all Euclidean transformations with fixed rotation of \( \theta \), that take \( a_i \) to within \( B^\epsilon \).
- a Euclidean transformation \( g \in E_2 \) such that \( \overrightarrow{d_H}(g(A), B) \leq \epsilon \) can be found if and only if \( \exists \theta \) such that

\[
S(\theta) = \bigcap_{i=1}^{m} B_i^\epsilon(\theta)
\]

is non-empty.
- Need to find \( \theta \) such that \( S(\theta) \neq \emptyset \).
Point Patterns

- $B^\epsilon$ is a union of discs.
- For given $\theta$: $S(\theta) = \emptyset$ or the boundary of $S(\theta)$ consists of arcs of radius $\epsilon$.
- Let $A(\theta)$ be the overlay of the boundaries of the sets $B_i^\epsilon(\theta)$.

Depth of a point $p$: number of $B_i^\epsilon(\theta)$ that contain $p$.
- $S(\theta) \neq \emptyset \iff$ some vertex of $A(\theta)$ has depth $m$. 
- Sweep through transformation space from $\theta = 0$ to $2\pi$. Similar to line sweep but sweep plane through $3D$ space.

- Keep track of the depth of the vertices of $A(\theta)$ as $\theta$ changes.

- Depth of vertices change at double and triple events:

- $O(m^2n^2)$ double events:
  $O(m^2n^2)$ pairs of discs, each pair touch at most twice

- $O(m^3n^2)$ triple events. Creation of Voronoi vertex of union of 3 transformed copies of $B$ in transformation space. At most $O(n^2k^2 \log^* k)$. $k = 3$, $O(m^3)$ ways to choose sets.
Algorithm

- Determine event points.
- Sort event points.
- Compute depths of vertices of \( A(0) \).
- Store vertices with depths in balanced binary tree.
- Triple event:
  at most 3 vertices either increment or decrement depth by 1.
- Double event:
  Create or delete 2 vertices. Upon creation initialise depth based on neighbouring vertices on same circle. If no neighbours, depth is 2.
- If any vertex has depth \( m \): Stop, output “yes”. Otherwise, output “no” after last event point.
- Algorithm dominated by sorting \( O(m^3n^2) \) event points. Thus runtime is \( O(m^3n^2 \log mn) \).
Line Segments

- Similar approach as for point patterns
- $B_i^\varepsilon$ not simply union of discs:
  Need to keep track of more information.
- Keep track of the boundaries of all the objects that make up each $B_i^\varepsilon$.
- An arc on the outer boundary of $B_i^\varepsilon$ is a portion of a circle generated by endpoint from $B$ and endpoint from $A$. $O(mn)$ such circles.
- A line segment on the outer boundary of $B_i^\varepsilon$ is generated by endpoint or segment from $B$ and endpoint or segment from $A$. $O(mn)$ such segments.
- $W$ is set of all $O(mn)$ such circles or line segments.
- Useful boundaries in $W$: outer boundary of some $B_i^\varepsilon$:
  label to indicate $i$ and keep track of which side of boundary lies inside $B_i^\varepsilon$. 
Algorithm

- Construct $W$ for $\theta = 0$.
- Determine coverage of each region ($O(m^2n^2 \log mn)$ time).
- Again consider double and triple events:
  - at most $\binom{mn}{2}$ double events
  - at most $\binom{mn}{3}$ triple events
  - can preprocess $W$ to find events.
- Sort events.
- Coverage information can be updated in constant time per event:
  When a new region is created examine an adjacent region:
  - if boundary is not useful, coverage is the same as for adjacent region.
  - if boundary is useful, coverage is one more or less than adjacent region.
• Need to maintain labeling of boundaries:
  – \( W_i \) is portion of \( W \) associated \( B_i^\epsilon \)
  – \( W_i \) has \( O(n) \) circles and line segments
    \( O(n^3) \) double and triple events for \( W_i \) as \( \theta \) changes.
  – Easy to keep track of usefulness of boundaries of \( W_i \).

• Each boundary portion of \( W \) is subset of a boundary from some \( W_i \).

• link boundaries in \( W \) with corresponding boundaries in \( W_i \).
  – maintain usefulness information in \( W_i \)
    total of \( O(mn^3) \) updates.
  – update linking information for each double or triple event of \( W \).

• \( O(m^3n^3) \) update events for \( W \).

• \( O(mn^3) \) update events over all \( W_i \).

• Must sort these events to get ordering of updates correct.

• Thus algorithm is \( O(m^3n^3 \log mn) \) overall.
Solid triangles and polygons

- Same method used for line segments can be used for solid triangles: only need to consider line segments that make up boundaries.
- Can triangulate polygons, and use same algorithm
References