Advanced Data Structures

Amalia Duch

CS-UPC

March 12, 2020

Kinetic Data Structures: Definition

This area of study has been introduced by Leonidas J. Guibas in 1998 [1].

A **Kinetic data structure** (KDS) is a data structure that stores data in motion, that is, data that change in some predictable way over time.

Kinetic Data Structures: Settings

Data: Each data point has a value that is a known function of time. For instance, we could have affine data ($x(t) = a + bt$).

Kinetic Data Structures: Settings

Operations: A KDS must support 3 types of operations:

- **modify($x, f(t)$):** The previous position function describing point x is replaced by $f(t)$. This allows us to refit the structure to mere up-to-date information, which is essential if the function used to approximate the position of a point is imperfect.
- **advance(t):** Advances the current time in the DS to t. The current time must be less than t.
- **query:** Queries the data structure relative to the current time. The specific queries are dependent on the DS in question.

Kinetic Data Structures: Open Problems

Deal rigorously and efficiently with the $modify$ operation.

Certificates: Definition

A *Certificate* is a testable logical statement pertaining to the DS. A set of certificates can provide a proof that a specific property of a DS is true.

Certificates: Example

The set of certificates:

- a is left to bc
- b is right to ad
- c is left to ad
- d is left to bc

provide a complete certification on the convex hull of abcd, as at least one certificate will change over time, the convex hull of the points changes its defining points also.

KDS: Approach

The basic idea for KDSs is to store a static data structure and the conditions under which the DS is valid, then, fix the Ds whenever the conditions are violated. So the approach consists of:

1. Store a DS that is accurate now (at the current time).
2. Augment the DS with a set of certificates sufficient to prove the DS validity.
3. Compute the failure times of each certificate.
4. Store the failure times in a priority queue.
5. As certificates fail, fix the DS and replace the certificates as needed.

KDS: Metrics

1. Responsiveness: When an event happens (e.g. a certificate fails) how quickly can the DS be fixed?
2. Locality: What is the maximum number of certificates any object participates in?
3. Compactness: What is the total number of certificates?
4. Efficiency: What's the worst-case number of events to be processed? Or What is the ratio of worst-case number of events processed to the worst-case number of ”necessary” changes (depends on the specific problem).

Example: Kinetic Predecessor

We want to support queries asking for the predecessor of an element (assuming the elements are sorted by value).
We take the following approach:

1. Maintain a binary search tree.
2. Let $x_1, x_2, \ldots, x_n$ be the in-order traversal of the BST. Keep the certificates $\{(x_i < x_{i+1}) | i = 1, \ldots, n-1\}$.
3. Compute the failure time of each certificate as:

$$failure\_time\_i := \min\{t \geq t_{now} x_i(t) > x_{i+1}(t)\}$$

4. Implement $advance(t)$ as follows:

Example: Kinetic Predecessor: $advance(t)$

```
while (t >= Q.front()) {
     t_now = Q.front();
     event(t_now);
     Q.pop();
 }
 t_now = t;
 ...
void event (x[i+1] <= x[i]) {
 swap(x[i],x[i+1],BST);
 add_certificate(x[i+1] <= x[i]);
 replace_certificate(x[i-1] <= x[i],
           x[i-1] <= x[i+1]);
 replace_certificate(x[i+1] <= x[i+2],
           x[i] <= x[i+2]);
}
```

Kinetic Predecessor Analysis

1. Responsiveness: Because we use a balanced BST and a certificate failure only affects $O(1)$ elements, we can fix the DS in $O(\log n)$ time.
2. Locality: Every object only participates in $O(1)$ certificates (in fact, always in at most 2).
3. Compactness: There are $O(n)$ total number of certificates.
4. Efficiency: Assuming ”pseudo-algebraic” motion (every pair of objects only changes order $O(1)$ times) the total number of DS events is $O(n^2)$, since there are n^2 pairs. If we have $\frac{n}{2}$ of points moving linearly with speed 1 and $\frac{n}{2}$ moving linearly with speed -1, they will change order $\Omega(n^2)$ times so the number of necessary events is $\Omega(n^2)$ and therefore the efficiency is $O(1)$.

Pseudo-Algebraic Motion

Motion is considered to be ”pseudo-algebraic” if all certificates of interest flip between true and false $O(1)$ times as the objects move.

Pseudo-Algebraic Motion is the required condition for Kinetic Data Structures to run efficiently.

Julien Basch, Leonidas J. Guibas, and John Hershberger.
Data structures for mobile data.
J. Algorithms, 31(1):1–28, 1999.