# Retroactive Data Structures

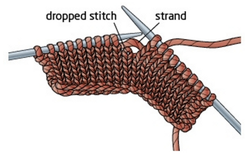
Michael T. Goodrich, Joseph A. Simons

Department of Computer Science, University of California, Irvine

## Motivation

We may need to alter the historical sequence of transactions.

- **Dynamization.** Some static algorithms require a dynamic data structure to process the input. These algorithms can be made dynamic by using a retroactive data structure instead.
- **Bad Data.** Data is either missing, was entered incorrectly, or comes from a tainted source (e.g. a sensor has malfunctioned).
- **Efficient Adaptation.** We may want to undo an error made early on without redoing all the work since then.




**Goal:** Maintain a large amount of data which changes over time such that we can efficiently update and query the data. Support updates and queries on current and past versions of the data structure.

## Definition

Dynamic data structures only allow us to make changes to the current state of the data structure.

- A data structure is **partially retroactive** if it supports updates to past versions of a data structure in addition to queries and updates to the current version.
- A data structure is **fully retroactive** if it also supports queries on past versions. We will focus on fully retroactive data structures.

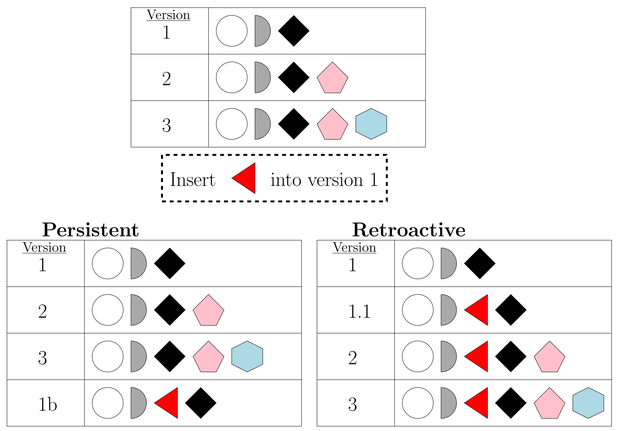



Figure: All later versions reflect the update in a fully retroactive data structure.

- *Unlimited Undo* or *Backtracking* is an inefficient approach. We want to find more efficient and elegant solutions.

## Retroactive Successor Queries

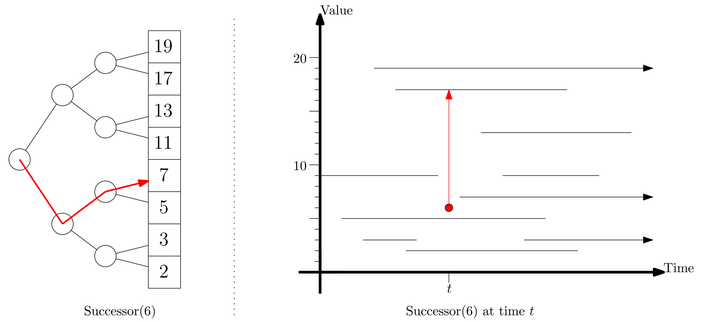



Figure: In retroactive successor queries, we can visualize the data points as horizontal line segments. Queries therefore reduce to dynamic vertical ray shooting.

- $m$ segments implies at most $2m$ rays.
- Each ray intersects the interior of two line segments.
- Expected number of rays intersecting randomly chosen segment is 4.

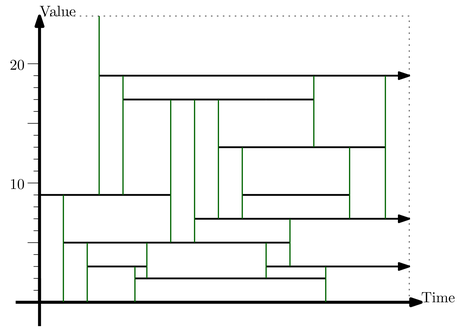



Figure: Trapezoidal Decomposition: Shoot vertical rays (green) from the endpoints of each line segment (black). This creates a rectangular subdivision.

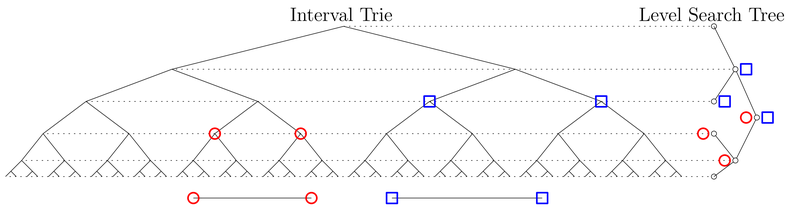



Figure: Stratified Tree. Intervals are stored at the highest level they cut, and at one node per level on the path to the root of the level search tree.

- Each $x$ interval defines a set of rectangles.
- Build stratefied tree on $y$ sides of rectangles.
- Query time: $\log \log^2(U)$

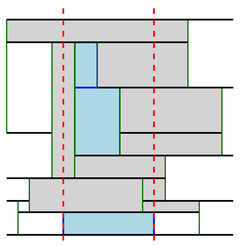
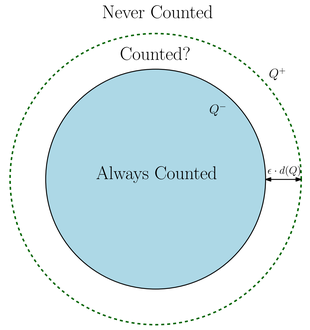


Figure: Support 2D point location using two layers of stratified tree.

## Approximate Range Searching




- Input: set of points in $\mathbb{R}^d$.
- Updates can insert or delete points at any time.
- Query: return points in approximate range at time $t$.

Figure: Points within $Q^-$ must be returned. Points outside $Q^+$ are never returned. Points between $Q^-$ and $Q^+$ may or may not be returned.

Two-layer data structure: Layer for time and layer for space.

**Two Ideas:**

**Time in Outer Layer**

- Treat data points as line segments parallel to time axis.
- Build segment tree on line segments
- Augment nodes of segment tree with dynamic approximate range searching structures, e.g. Quadtreap or Skip-Quadtree.
- Speed up inner queries with fractional cascading techniques.

**Space in Outer Layer**

- Use a tree with few rotations, e.g. weight-balanced B-tree
- Augment each leaf with update times.
- Internal nodes store union of children's times.

  **Key property**: parent not present $\implies$ children are not present.

Two assumptions to augment with stratefied tree

- Update times are integers (fixed universe, size $m^k$).
- Updates are given in random order.

Cell Query and Expected Update Time: $O(\log n \log \log n)$

[1] **Berg, M. D., Kreveld, M. V., and Snoeyink, J.** Two- and three-dimensional point location in rectangular subdivisions. *Journal of Algorithms 18* (1995), 256–277.

[2] **Demaine, E. D., Iacono, J., and Langerman, S.** Retroactive data structures. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2004), SODA '04, Society for Industrial and Applied Mathematics, pp. 281–290.

[3] **Eppstein, D., Goodrich, M. T., and Sun, J. Z.** The skip quadtree: a simple dynamic data structure for multidimensional data. In *Proceedings of the twenty-first annual symposium on Computational geometry* (New York, NY, USA, 2005), SCG '05, ACM, pp. 296–305.

[4] **Mount, D. M., and Park, E.** A dynamic data structure for approximate range searching. In *Proceedings of the 2010 annual symposium on Computational geometry* (New York, NY, USA, 2010), SoCG '10, ACM, pp. 247–256.

[5] **Seidel, R.** A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl 1* (1991), 51–64.

University of California · Irvine

Center for Algorithms and Theory of Computation
Donald Bren School of Information and Computer Sciences
University of California, Irvine

Supported by: Office of Naval Research award N00014-08-1-1015.

Scalable Methods for the Analysis of Network-Based Data