Set-Difference Range Queries

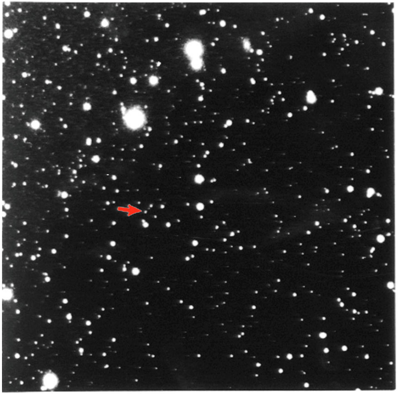
David Eppstein, Michael T. Goodrich, and Joseph A. Simons

25th Canadian Conference on Computational Geometry
Waterloo, Ontario, August 2013

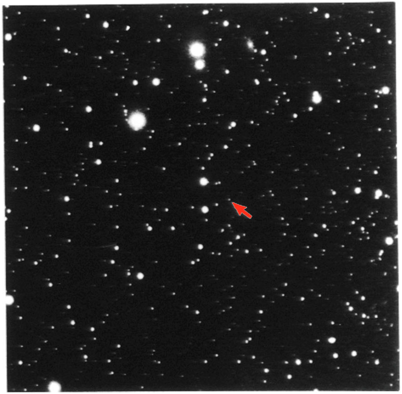
Spot the difference

A popular type of children's puzzle

The discovery of Pluto



January 23, 1930



January 29, 1930

The original plates from Clyde Tombaugh's discovery of Pluto, recolored to make the arrow markers more obvious

Detecting alterations of photographic images

Kliment Voroshilov, Vyacheslav Molotov,
Joseph Stalin, and Nikolai Yezhov, 1937

Before

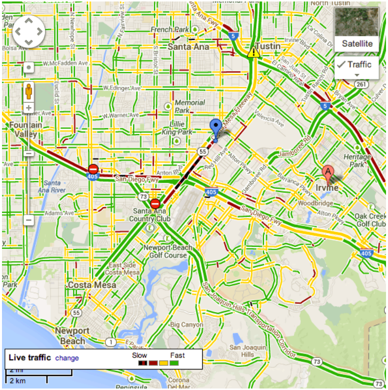
After

Local differencing

Find differences within a restricted subset of the input
(e.g. to avoid getting distracted by bigger differences elsewhere)

Non-imaging applications of local differencing

- Synchronize calendars for a range of dates
- Reconcile a range of database transactions
- Find variant DNA in one or more genes of a genome
- Track a small set of moving objects among many non-moving objects
- Communicate updated data for a windowed view of a road map



Google Maps live traffic display near Orange County Airport, California, 2013-08-01 16:30

Our contributions

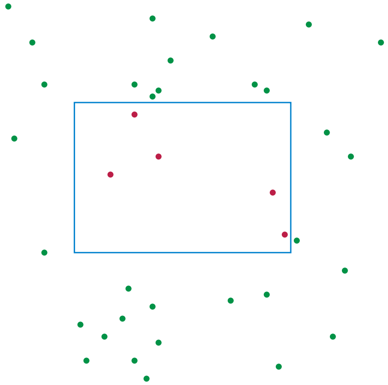
Main contribution: Formulate **set difference range querying**, a formalization of the local differencing problem within the framework of range query data structures

Secondary contribution: Combine known data structural techniques for decomposition of range queries and for streaming straggler detection to **solve set difference range queries efficiently**

Range spaces

A range space consists of

- A family of **objects** parameterized by $O(1)$ real numbers
- A family of **ranges** parameterized by $O(1)$ real numbers
- An **incidence relation** between objects and ranges



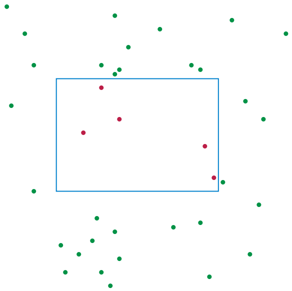
e.g. points in the plane, rectangles, containment

Range querying

Input: **finite set of objects** from a range space, a value for each object, and an (associative, commutative) aggregation operator

Preprocessing: construct a space-efficient **data structure**

Query: find points in a query range and return their **aggregate value**



To count points in range: value = 1, aggregate = addition
To find top priority point: value = priority, aggregate = max
To list all points in range: value = self, aggregate = union

Set difference range queries

Data: One or more sets of objects

Object values = members of some universe of sets

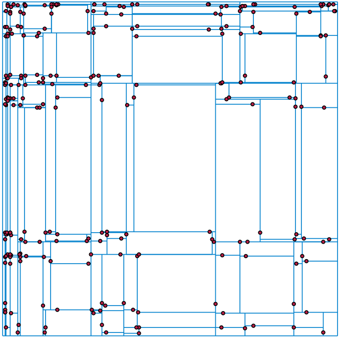
Query: two ranges (possibly in different sets of objects)

Aggregation: Elements that belong to one range but not both
(symmetric difference of sets)

Canonical ranges

Standard strategy for range query problems:

- Identify a small set of *canonical ranges*
- Store the aggregate value of each canonical range
- Decompose query ranges into few canonical ranges

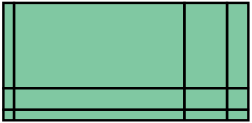


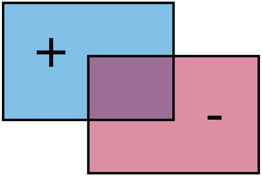
Example: kD-tree

$O(n)$ canonical rectangles

Query rectangle decomposes into $O(\sqrt{n})$ canonical rects

Group vs semigroup models





Semigroup: query decomposed into **disjoint** canonical ranges

Can be combined using only the aggregate operator

Allows more general types of aggregation

Group: query decomposed into **overlapping** canonical ranges

Inclusion-exclusion formula using subtraction

Allows more general types of decomposition

Set differencing in the group model

Instead of sets, use *multisets*:
integer counts of how many times each element appears

The members of a multiset are the elements with nonzero counts

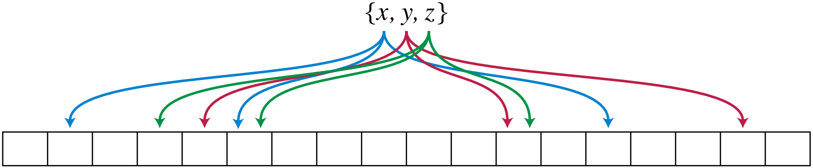
Vectors of counts can be added and subtracted

The set difference is just the subtraction of two vectors

PD image File:Tally marks counting visitors.jpg by Achird on Wikimedia commons

Invertible Bloom filters

[E & Goodrich, WADS 2007 & IEEE TKDE 2011]



Hash each element to $O(1)$ cells of a table, #cells = O(capacity)

Each cell stores $\sum$elements, #elements, checksum

Can add/subtract multisets of arbitrary size
(by adding/subtracting values in each cell)

Decode by finding cells containing only one element,
possible whenever size of result $\leq$ capacity

How to perform set-difference range queries

- Construct a family of canonical sets
- Decorate each set with invertible Bloom filters of capacities $1, 2, 4, \ldots$ set size
- To handle a query:
  - Decompose into canonical ranges
  - For capacity $= 1, 2, 4, \ldots$, add/subtract canonical IBFs to construct an IBF for the difference of the two query ranges
  - When capacity is large enough for the resulting IBF to be successfully decoded, stop and return the result

U.S. Navy photo 050215-N-2636M-015, Nick Leones, by Kleynia McKnight

Analysis

Space = input size × number of canonical sets per object, similar to other typical range query data structures

(Slightly more space-efficient if output size fixed in advance)

Query time = output size × number of canonical sets per query

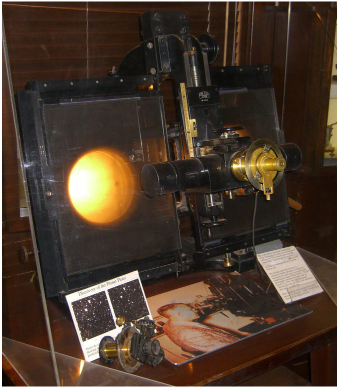
Can also be modified to return approximate cardinality of result, with query time polylog × number of canonical sets

(Uses frequency moment estimation sketch in place of IBFs)

Conclusions

New, natural and useful range querying problem

Efficient solutions, independent of the exact shape of the ranges, that can be combined with most other range querying techniques



The blink comparator used by Clyde Tombaugh to discover Pluto