Donald Whyte
sc10dw

Accelerating Multi-Dimensional Search

Student-Supervisor-Assessor FYP Project Meeting

(16/04/14)

Requirements

1. Produce literature review describing the current state of multi-dimensional search structures, comparing their strengths and weaknesses and highlighting core challenges of the field
2. Implement at least one index structure and perform performance analysis of the implementation
3. Perform one set of modifications to the implemented structure in an attempt to optimise its performance
4. Evaluate and compare performance of final implementations to pre-defined baselines, unoptimised structures and their reported performance in the literature

Milestones Met

- Project Outline Defined
- Literate Review Complete
- Test Data Collected
- Mid-project Presentation Delivered
- Initial Implementation of Software Deliverables Complete

Milestones Remaining

- Software Deliverables Finished
- Final Report Finished
- Student Symposium Delivered

Donald Whyte
sc10dw

Changes from Original Plan

- Splay quadtree was not implemented due to lack of detail in some of the algorithms
  - could have filled in the blanks, but this meant developing entirely new extensions to the algorithms (which adds risk)
- Greater time exploring the Pyramid tree and variants of it than expected, due to good performance (relative to baselines) on most datasets
- Weekly iterations ended up taking two weeks rather than one, due to a greater number of structures/variants being developed and analysed in each iteration (more to discuss/explore than expected!)

Main Structures Developed

Structure	**Description**
Pyramid Tree	As described by Berchtold et. Al in 1998 paper, except using hash table instead of B+-tree
Pseudo-Pyramid Tree	Simplified version of Pyramid tree developed by Zhao Geng. Uses a point's distance from each dimensional boundary to compute a hash value for the point
Pyramid Tree Variants	• Index-based Pyramid trees • Bucket pyramid tree • Splay pyramid tree
Hash Table	Uses boost's hashing function to give almost-unique hash to every point
KD-Tree	Simple kd-tree implementation where dimensions are cycled through at each level as cutting dimension. Single point at each node

Core Iterations

- **Iteration 1** – baselines, unoptimised pseudo-pyramid tree, general optimisation techniques such as using compiler optimisation
- **Iteration 2** – accelerating pseudo-pyramid tree, pyramid tree, hash table and basic kd-tree
- **Iteration 3** – kd-tree modifications/variants, trying to accelerate it for point queries further and improving individual point remove

Donald Whyte
sc10dw

Evaluation Approach

Performance of three operations being analysed:

- insert
- delete
- point query

Rather than timing an individual operation, the total execution time of multiple invocations of an operation will be timed, using different points. Two main types of operation lists have been used:

Operation List	**Description**	**What's Measured**
Insert-Query-Delete	Inserts all points in dataset, then queries each point and finally deletes each point	Total execution time for each **operation type**
Random Operations	Contains randomly generate operations using points taken from the target dataset and points not present in the dataset.	Total execution time of all operations
Real Program Traces	Traces of real insertion/clear operations used to construct joint contour nets. Provided by David.	Total execution time of all operations

Why?

- **Insert-Query-Delete** provides a controlled test for how each operation performs isolation
- **Random Operations** is being used to measure how well the structures perform in scenarios that may occur if the structure will be used in an application, as it contains interleaved operations and queries with non-existent points.
- **Real Program Traces** allow the structures to be evaluation with respect to how they perform in a known real application, to ensure that the results gained from the synthetic operation lists are consistent (or if they differ, why).

Test Datasets

Dataset	**Operation Lists**	**Varying Parameter**
Small Synthetic Data • Uniform Distribution • Skewed Distribution • Clustered Distribution 10,000 Points	Insert-Query-Delete	Dimensionality

Large Synthetic Data (16D) • Uniform Distribution	Insert-Query-Delete	# of Points (randomly sampled)
Astrophysics (10D)	Insert-Query-Delete Random Operations	# of Points (randomly sampled)
Hurricane Isabel (13D)	Insert-Query-Delete Random Operations	# of Points (randomly sampled)
Geometric Mesh (3D)	Insert-Query-Delete Random Operations	---
Joint Contour Net Construction Trace (3D) (Slab Width 4, Active)	Trace of Real Program (Insert/Clear)	---
Joint Contour Net Construction Trace (3D) (Slab Width 4, Centrals)	Trace of Real Program (Insert/Clear)	---

Performance Summary of Insertion/Point Queries

Sequential scan performs worse than any other structure and the octree cannot be tested beyond 10 dimensions or a larger number of points due to it running out of memory.

Impact of distribution:

- While the distribution of the synthetic data had a small impact on the performance, the differences are negligible with all tested structures
- However, the scientific computation datasets (astrophysics and hurricane Isabel) have large regions of uniform/similar points (i.e. data sparsity)
    - causes Pyramid tree variants to drastically slow down, to as slow as 90 seconds for 500,000 points! (compared to ~0.5 seconds for kd-tree)

Impact of dimensionality:

- Sequential scan does not perform much slower when dimensions increase, but the octree quickly becomes unusable due to exponential memory requirements
- As the number of dimensions increase, the Pyramid tree's performance is reduced, but at an extremely slow rate (making it usable for dimensions as high as 200)
    - this also applies to the hash table
- Despite the background literature's findings on kd-tree performing poorly in higher-dimensional spaces, for *point queries*, the performance of kd-trees is not impacted much, with only a slightly decrease in performance

Impact of dataset size:

- Up to 50,000 points, the kd-tree typically performs the fastest, followed by the Pyramid tree variants and then the hash table
- As the number of points increase beyond this, the hash table begins to get faster and faster than the kd-tree
    - same with the pyramid tree variants, with the exception of the scientific datasets

Impact of operation order (operation lists):

- Results are consistent across Insert-Query-Delete and Random Operations lists
- For the joint contour net real program trace, the best performing structure varied:
    - on the *active* trace, kd-tree was the fastest, followed by the hash table and then the pyramid tree variants
    - on the centres trace, hash table was the fastest, followed by pyramid tree variants and the kd-tree
    - reason for this is that the *active* dataset has a larger number of clears, with less points being stored in the structure at any one time than the centers dataset
    - all hash-based approaches take longer to re-initialise (i.e. clear) than the kd-tree, and hash-based approaches only become faster than the kd-tree with a larger number of points

Performance Summary of Point Deletion

kd-tree consistently has the slowest point deletion, regardless of dataset.

Impact of distribution:

- for all synthetic data, the pyramid tree provides the fastest remove, followed by hash table

Impact of dimensionality:

- dimensionality increases removal time since it typically takes longer to locate the point to remove in all structures. This is the same reason insert/point queries are slower in higher dimensions. The general relationship is, as you add x dimensions, the time it takes to perform the same amount of removals (ignoring data distribution) is multiplied by $2x$

Impact of dataset size:

- more data means removals take longer as more generally needs to be searched

Donald Whyte
sc10dw

Tables and Plots for Final Report

For each of the three iterations, the following tables will be provided:

- insert/delete/query timings for small synthetic data (varying dimension)
- insert/delete/query timings for large synthetic data (varying size)
- insert/delete/query timings for astrophysics data (varying size)
- insert/delete/query timings for hurricane Isabel data (varying size)
- total timings for JCN construction traces (varying size)
- max heap memory used and cache hit rate for astrophysics data (varying size)

For each of the three iterations, the following plots will be provided:

- insert/delete/query timings for small synthetic data (varying dimension)
- insert/delete/query timings for large synthetic data (varying size)
- insert/delete/query timings for astrophysics data (varying size)

Structure	**Operation**	**1D**	**2D**	**3D**	**10D**	**50D**	**100D**
Sequential Scan	Insert Delete Point Query	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000
Octree	Insert Delete Point Query	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000
kd-tree	Insert Delete Point Query	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000
Pyramid Tree	Insert Delete Point Query	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000
Hash Table	Insert Delete Point Query	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000	0.0000 0.0000 0.0000

Donald Whyte
sc10dw

Structure	**Max Heap Memory (MB)**	**Cache Hit Rate**	**Dominant Function *(% Time Spent)***
Sequential Scan	24.45	10.486%	Point::operator== *(34.3%)*
Octree	60.36	63.22%	Octree::subdivide() *(75.9%)*

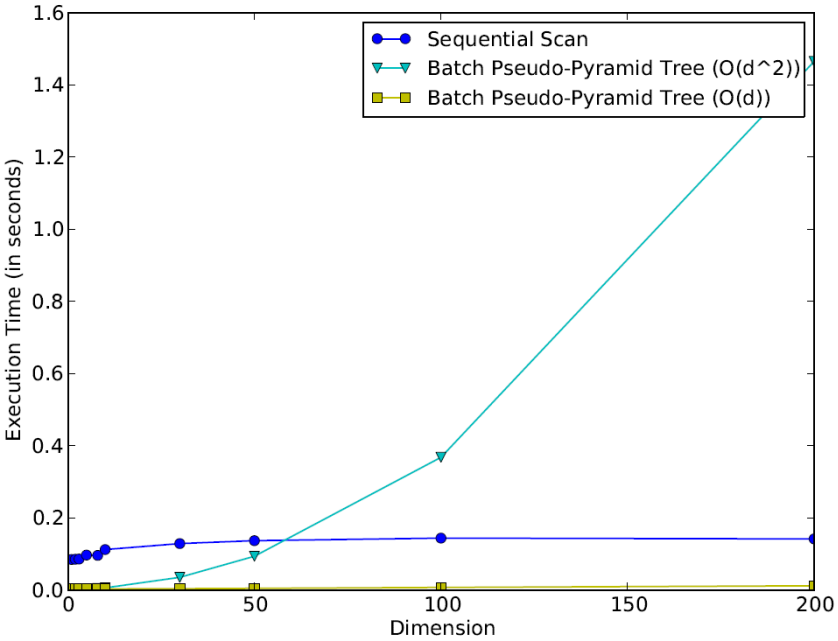

Illustration 1: Total Execution Time of 10,000 Insert Operations on Uniformly Distributed Synthetic Data

Donald Whyte
sc10dw

General Report Structure

See draft table of contents attached to back of sheet.

Future Work/What's Next

- Further exploration of the kd-tree, developing a few small optimisations and variant and comparing them
- Generate full timing tables and plots for each iteration and integrate into report
- Final report write up
  - finish remaining content for iteration 2 section
  - iteration 3 section
  - revised methodology section, conclusion
  - tweaks, modifications, etc., etc.