Parallel kd-tree with Batch Updates

Ziyang Men, Zheqi Shen,
Yan Gu, Yihan Sun

Slides by Younghun Roh

Historical Context

Paper @ SIGMOD

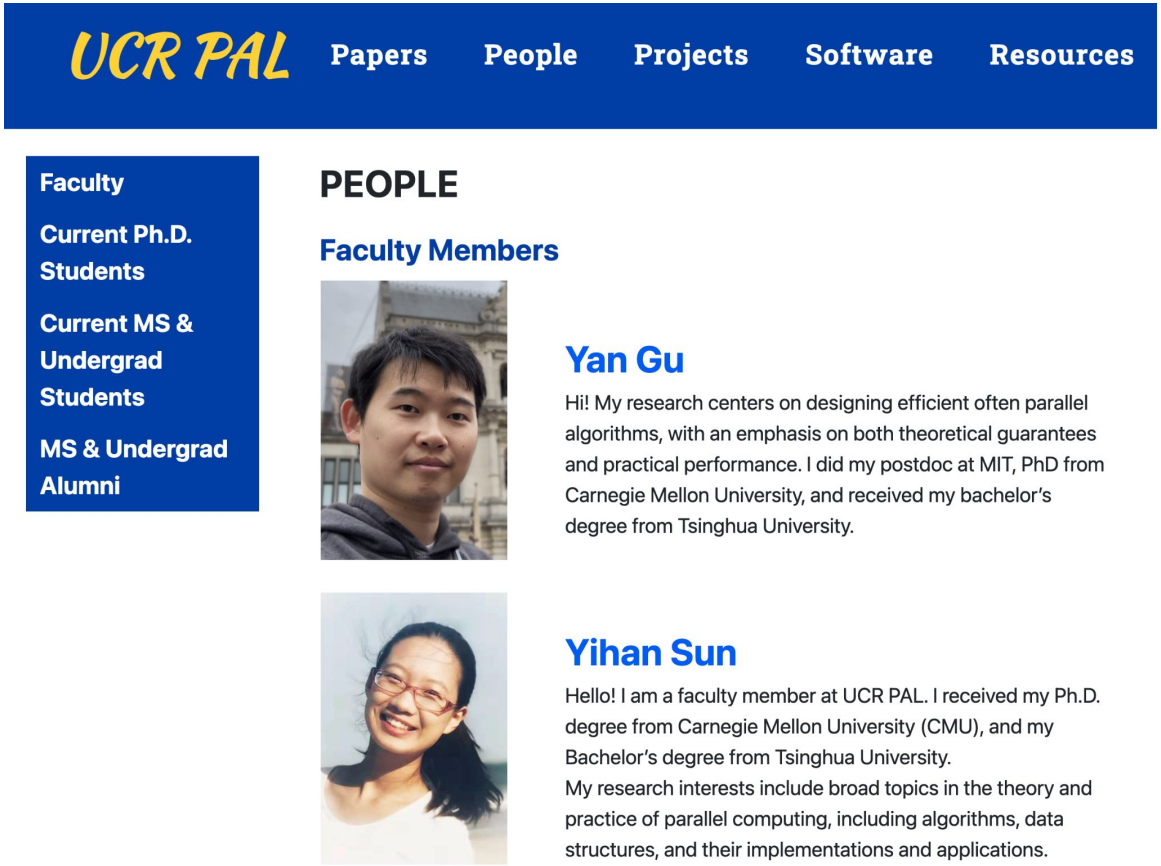
Ziyang Men

Zheqi Shen

Yan Gu (Postdoc with Julian)

Yihan Sun (Married ^)

All from UC Riverside



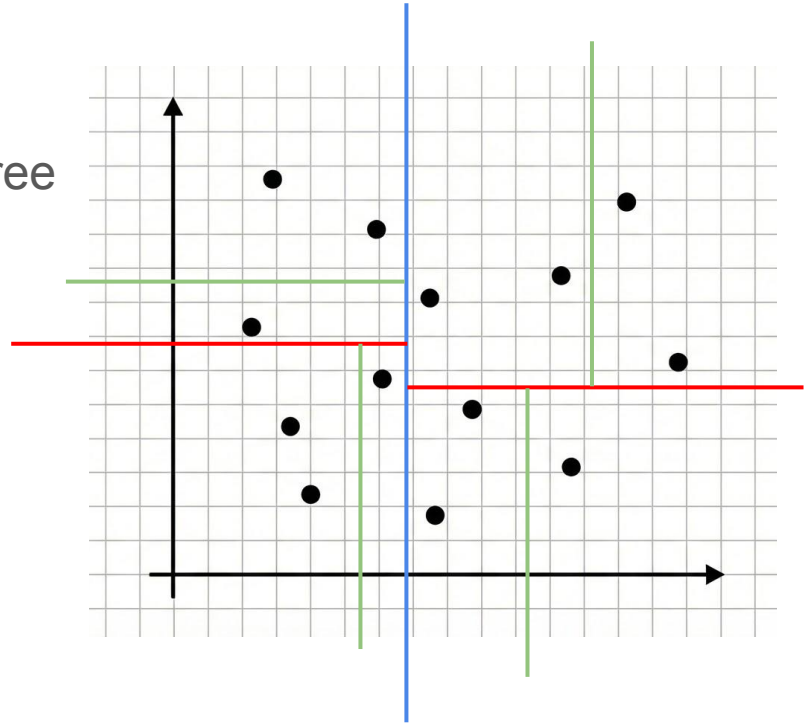
What is kd-tree?

Managing (k-dim) n points like Binary Search Tree

- For each node, cut the space in ‘half’
- Choose any dim, cut until base case (ϕ)

Operations

- Insert, Delete (= Updates)
- Queries
  - k Nearest Neighbors, range find / count, etc
- (Construct, *Balance*)



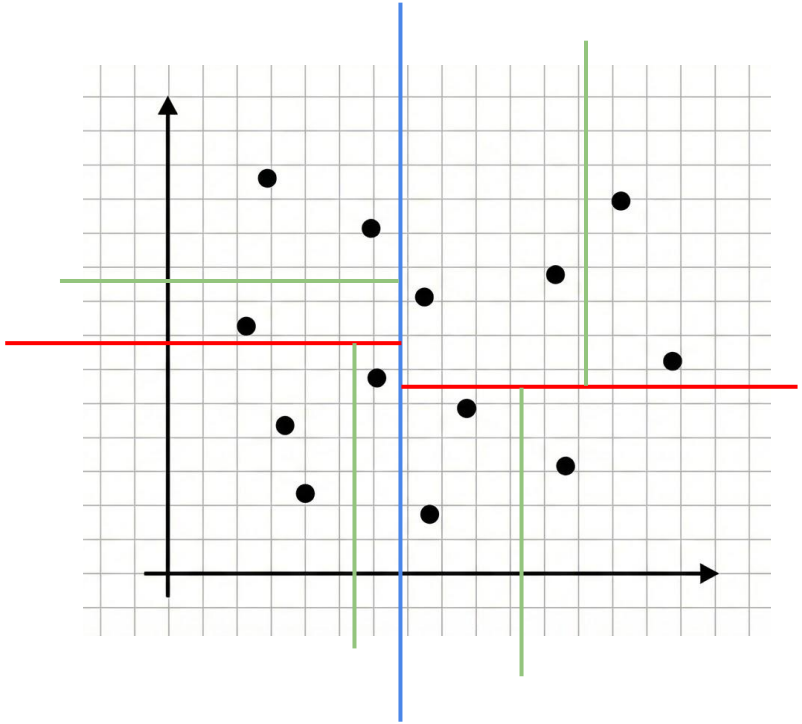
What is kd-tree?

Analysis (balanced, sequential)

- Dimension D <~ 10
- O(n) space, O(n log n) construction
- $O(\log^2 n)$ update, $O(n^{(1-1/D)}+m)$ query

Applications

- Databases, Data science
- Machine learning, Clustering algorithms
- Computational geometry



Objective & Overview

Designing parallel kd-tree

- (Read-only) Queries can be parallel
- Updates are processed in batch of m ops

https://github.com/ucrparlay/Pkd-tree

Goal (with high probability)

- Work O(n log n), span O(polylog n)
- Cache complexity $O(n/B \log_M n)$ (i.e. almost O(n/B))

How?

- Construct faster; do multiple levels at once
- Always work in cache-sized chunks – avoid block movement

Preliminaries

- Points have different coordinates
  - For any dimension, we can split points in half
- Probabilistic algorithm
  - Algorithm has random sampling
  - Most analysis works ‘with high probability’ (w.h.p)
  - w.h.p. in terms of n means: 1 - n^{-c} for any c > 0
- D is small (~10)

T	a (sub-)kd-tree, also the set of points in the tree		
ϕ	leaf wrap threshold (leaf size upper bound)		
k	required number of nearest neighbors in a query		
λ	number of levels in a tree sketch (i.e., that are built at a time)		
$\mathcal{T}$	tree skeleton at T with maximum levels λ		
P	input point set (for updates, P is the batch to be updated)		
$T.lc$	left child of T	$T.rc$	right child of T
n	tree size	m	batch size for batch updates
D	number of dimensions	d	a certain dimension
S	samples from P	s	size of the S
σ	oversampling rate	α	balancing parameter
M	small-memory (cache) size	B	block (cacheline) size

Table 2. Notations used in this paper.

Tree Structure Overview – ??

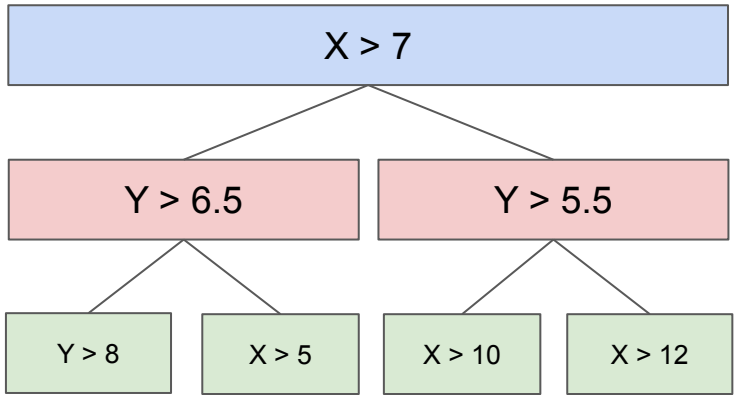
(Heap-like memory structure?)

Put points all in one array,
and internal nodes keep ranges and its children internals;
We re-arrange points when building tree.

Static queries are same. We don’t analyze here

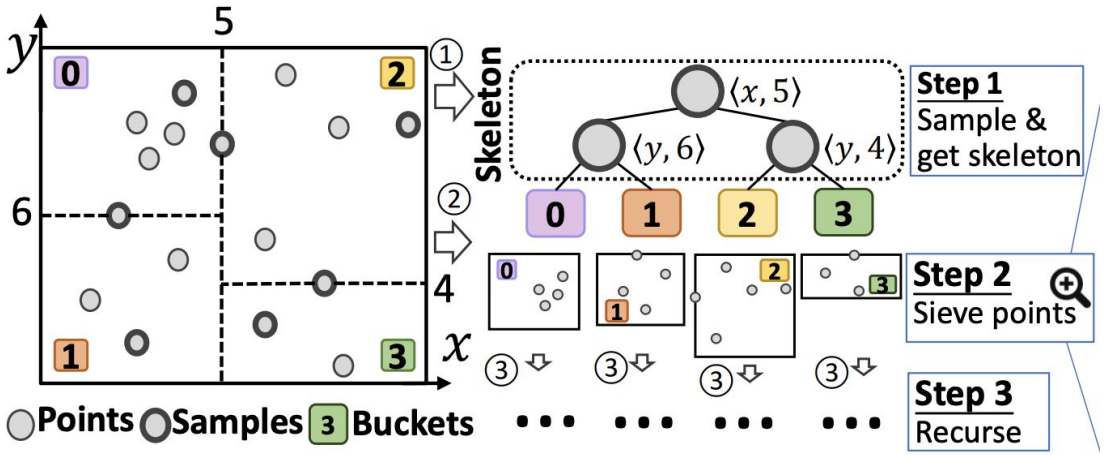
‘Plain Parallel kd-tree Construction’

- Do the below, and recurse down until base case
- Given array of n points, find a median and partition all points:
  - Parallel find a pivot in a chosen dimension
    - Divide and Conquer with Random sampling, or median of medians
  - Parallel partition in <, =, > (get prefix sum and relocate)
  - Recurse until base case
- O(n log n) work, $O(\log^2 n)$ span
  - O(log n) levels, O(n) work and O(log n) in each level
  - Bad cache efficiency near the root
    - O(log n) rounds of data movement



Parallel Construction Overview

- From n points **P**, get a sample of points **S**
- Build skeleton kd-tree with **λ** levels, with only **S**
- Distribute all **P** according to the skeleton tree (sieve points)
- Recurse until base case (ϕ)

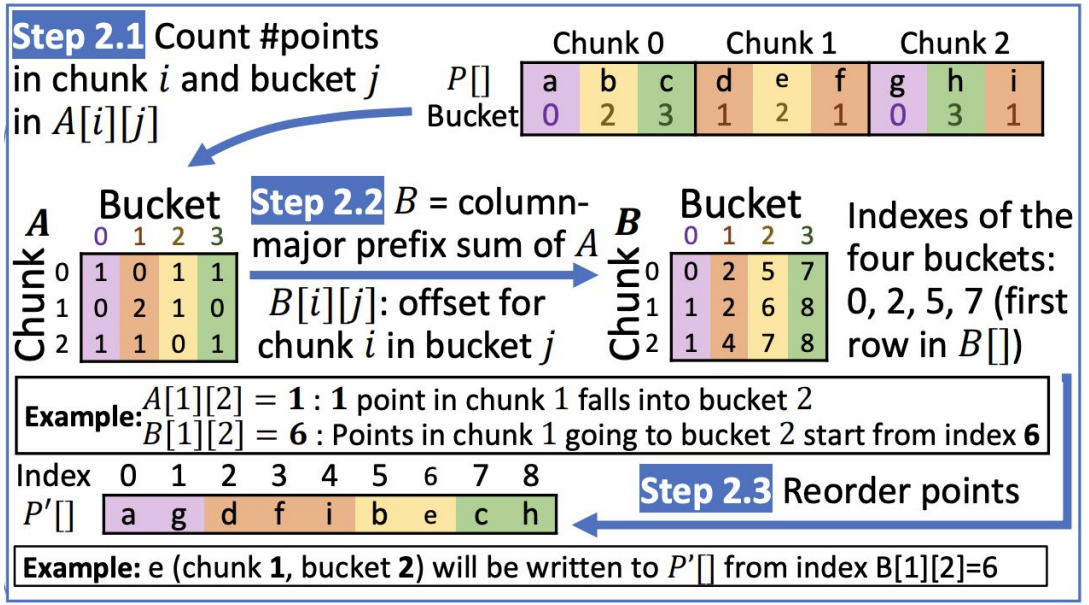


Sieve Step

We have **λ** levels (= 2^{λ} buckets) into which we distribute points

Each thread processes chunk with **L** points (fits in cache)

- Get bucket id for each point
- Build a count, and prefix sum
- Relocate points accordingly



Batch Update

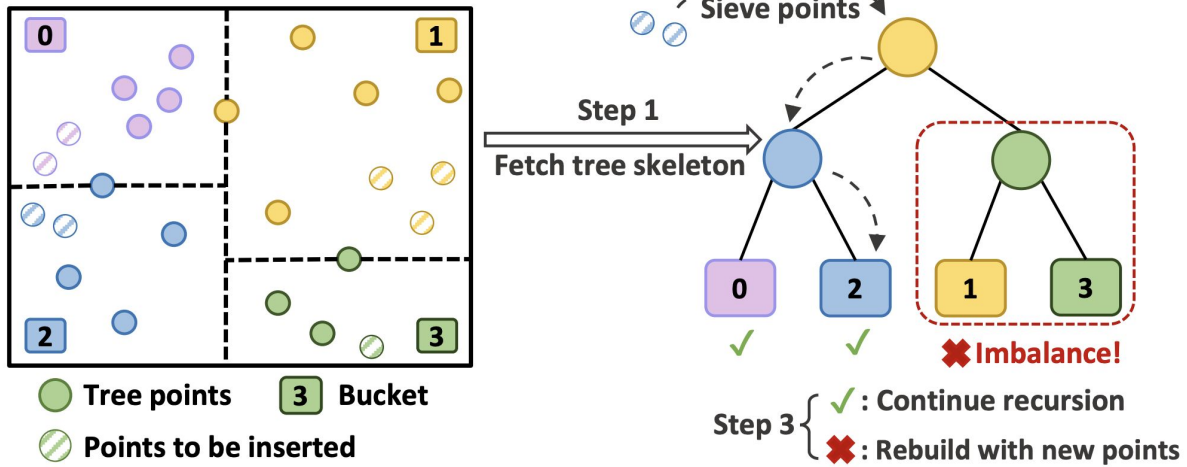
How to maintain balance while updating?

- Set a balancing margin **α**, and reconstruct when it's too imbalanced

Use the existing kd-tree as a skeleton, and sieve points

Reconstruct the whole subtree when it's imbalanced

Note: **λ** can start anywhere



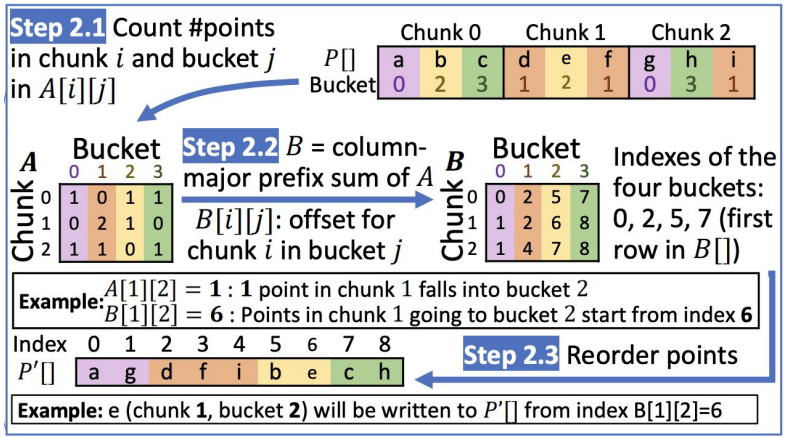
Analysis

What is a probability of skeleton (built from samples) being ‘even’?

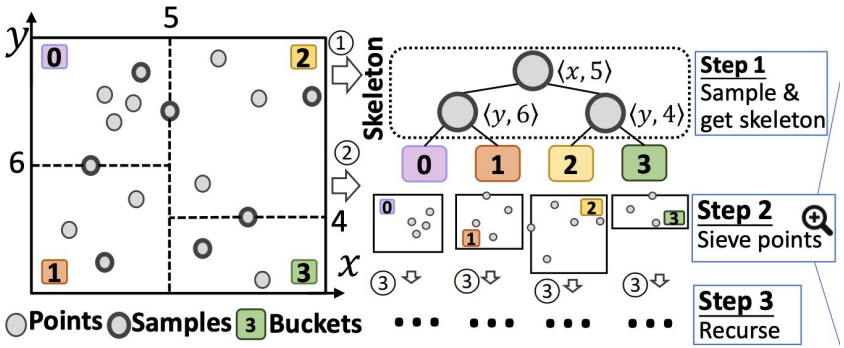
- Less points, higher ‘skewness’ (farther from the real median), deeper tree
- Sample size about 2^{λ} c log n is safe enough (why?)

Construction:

- O(n log n) work, O($\log^2$ n) span*
- All fit in cache, except copying points
- O(n/B) for each recursion,
- so O(n/B log n / λ) cache complexity



Parameters & Implementation



Parameters (decided from experiments)

- Levels λ = 6 (= ε log M), chunk size L = 2^{λ}
  - To do sieve inside cache
- Sampling size σ = 32, base case size ϕ = 32,
  - To find good enough skeleton; Granularity control
- Balancing parameter α = 0.3
  - Subtree size can be 0.2n - 0.8n – balancing depth and reconstruction
- Put points in array, and make internal nodes point to intervals
- Choose widest dimension for splitting

Experiments - Basic Benchmark

	Build				Batch Insert (1%)				Batch Delete (1%)				10-NN (1%)				Range Report (10K)			
D	2	3	5	9	2	3	5	9	2	3	5	9	2	3	5	9	2	3	5	9
Uniform-1000M																				
Ours	3.15	3.65	5.67	9.66	.104	.107	.123	.152	.121	.134	.171	.232	.381	.822	4.58	108	.391	.706	2.31	16.2
LOG	37.9	45.4	58.0	92.7	2.16	2.66	3.67	6.19	.396	.485	1.94	2.39	2.96	4.48	20.2	879	2.62	4.14	8.94	31.6
BHL	31.7	40.5	58.4	104	31.4	40.3	57.1	103	30.9	39.3	68.7	114	.487	1.02	7.38	448	2.06	2.94	6.53	23.2
CGAL	1147	1079	1217	1412	1660	1815	1863	2145	41.2	41.3	45.0	40.2	1.04	2.30	12.5	189	311	282	249	184
Varden-1000M																				
Ours	3.66	4.78	6.27	11.2	.055	.107	.157	.350	.049	.112	.127	.237	.172	.210	.336	.433	.382	.745	2.24	13.1
LOG	34.2	41.8	57.8	92.6	2.01	2.60	3.72	6.07	1.06	1.14	1.92	2.30	2.05	2.29	23.3	2225	2.63	4.25	7.95	14.1
BHL	30.2	39.2	58.7	104	29.4	39.1	57.3	102	29.0	38.4	67.0	123	.242	.324	.456	.535	1.95	3.03	5.72	9.96
CGAL	429	390	372	438	849	700	582	599	13.0	9.53	23.1	3.90	.511	.217	.318	.392	296	283	253	278

Table 3. Running time (in seconds) for Pkd-tree and other baselines. Lower is better. D: dimensions.

Experiments - Metrics

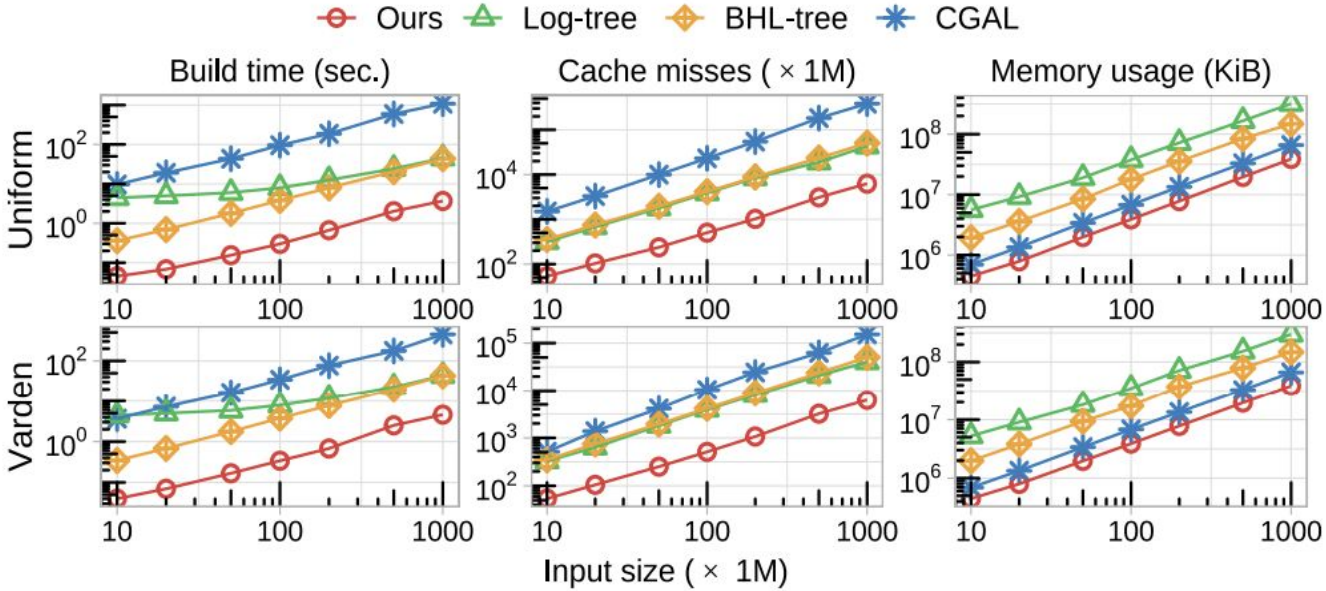


Fig. 8. Time, cache misses, and memory usage needed during the tree construction. Lower is better. Plots are in log-log scale. All points are in 3 dimensions.

Experiments - Real-World Benchmark

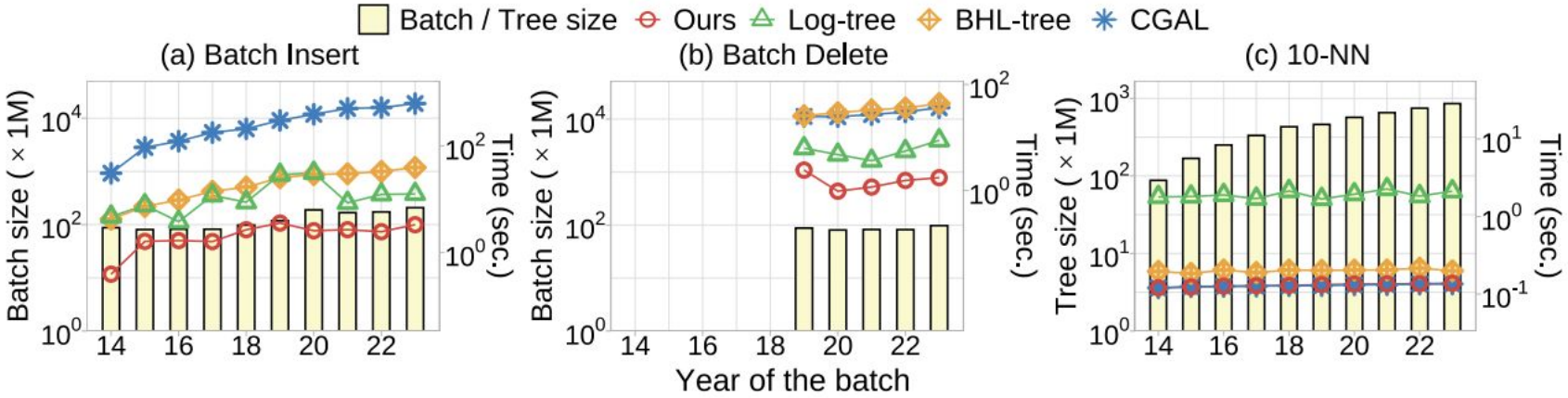


Fig. 7. Batch update using a sliding window spanning five years and 10-NN queries on OSM [45]. Lower is better. The input is batched by years from 2014 to 2023. In each year, we insert the corresponding

Conclusion

Parallel kd-tree with batch update - 10x faster

- Construction

THEOREM 3.4 (IMPROVED SPAN). *A Pkd-tree of size n can be built in optimal* $O(n \log n)$ *work and* $O(\text{Sort}(n)) = O((n/B) \log_M n)$ *cache complexity, and has* $O(\log^2 n)$ *span, all with high probability.*

- Batch update

THEOREM 4.1 (UPDATES). *Using* $\sigma = (6c \log n)/\alpha^2$, *the update (insertions or deletions) of a batch of size* $m = O(n)$ *on a Pkd-tree of size n has* $O(\log^2 n)$ *span* whp; *the amortized work and cache complexity per element in the batch is* $O(\log^2 n)$ *and* $O(\log(n/m) + (\log n \log_M n)/B)$ whp, *respectively.*

Review

Awesome writing

- Preliminaries/definitions, diagrams, codes, order of presentations, …

Comprehensive experiments

- Little too comprehensive – hard to track all observations and decisions

Rigorous complexity analysis

Is not always fastest?

- Why was it slow in some cases, esp. high D? (no bounding box / worse split?)

How to choose good parameters? (Are they universally good?)

Questions & Future Works

Can we parallelize single range report query?

In sieving, can we get bucket id in min(D, λ) time? (Instead of λ)

Can we avoid sieving with atomic operations?

- Instead of building prefix index, copy points with faa(bucket.index)

Can we make this deterministic?

Can we do similar things with usual segment/interval trees?

- Coalescing levels to make things cache-friendly