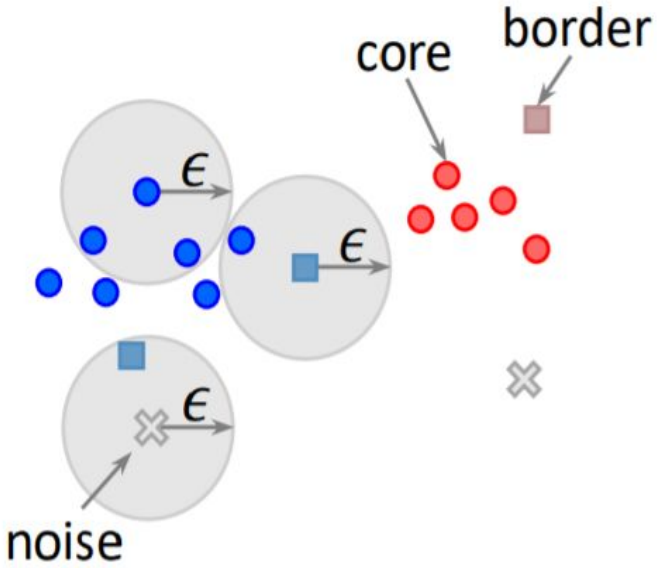
Theoretically-Efficient and Practical Parallel DBSCAN

Yiqiu Wang, Yan Gu, Julian Shun

DBSCAN (ε, minpts)

Minpts = 3



Previous work

- Parallel
  - Xu et al. PDBSCAN
    - Distributed R-Tree, exact answer
  - Patwary et al. PDSDBSCAN
    - Parallel lock based union-find, approximate without guarantee
  - Gotz et al. HPDBSCAN
    - Data partition, process locally and then merge results, exact answer
  - RP-DBSCAN
    - Distributed map-reduce, approximate without guarantee
  - ...
- Serial
  - Gunawan et al.
    - Sequential exact DBSCAN with theoretical guarantees
  - Gan and Tao
    - Sequential approximate DBSCAN with theoretical guarantees

Our contribution

- Practical parallel algorithms for 2D exact DBSCAN, and higher dimensional exact and approximate DBSCAN with work bounds matching the best sequential algorithms, and polylogarithmic depth.
- Highly-optimized implementations.
- A comprehensive experimental evaluation
  - 2-38x self-relative speedup (36 cores)
  - 5-33x speedup over best sequential (36 cores)
  - 14-6102x faster than existing parallel implementations
  - Processes largest dataset known for DBSCAN (> 4 billion points) on just one machine

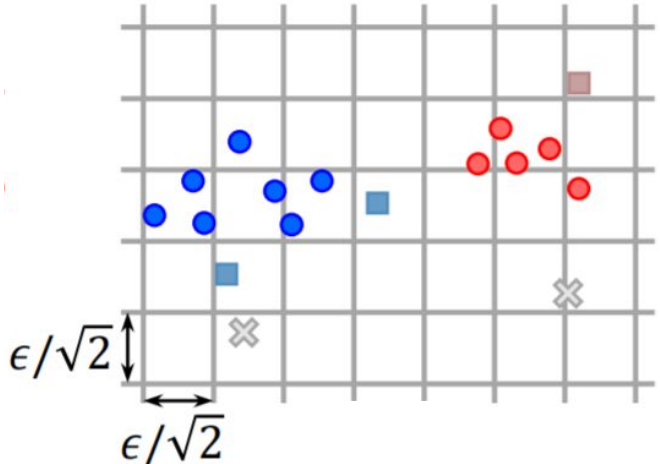
DBSCAN High-level Algorithm (2D)

- Construct cells
- Mark core points
- Cluster core points (cell graph)
- Cluster border points

DBSCAN High-level Algorithm (2D)

- Construct cells
- Mark core points
- Cluster core points (cell graph)
- Cluster border points

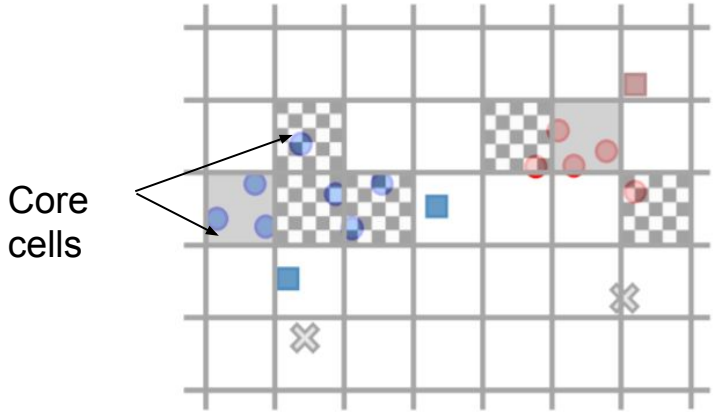
- Side length ε/√2
- O(n) work in expectation
  O(log n) depth w.h.p.



DBSCAN High-level Algorithm (2D)

- Construct cells
- Mark core points
- Cluster core points (cell graph)
- Cluster border points

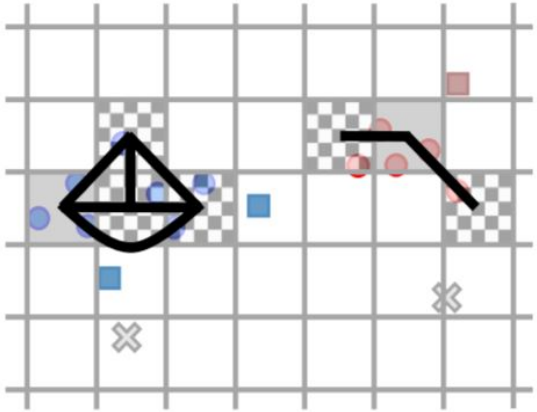
- For each cell
  - If |cell| > minpts(3): all are core points
  - Else: perform range count for each point
- O(n · minPts) work
  O(log n) depth w.h.p.

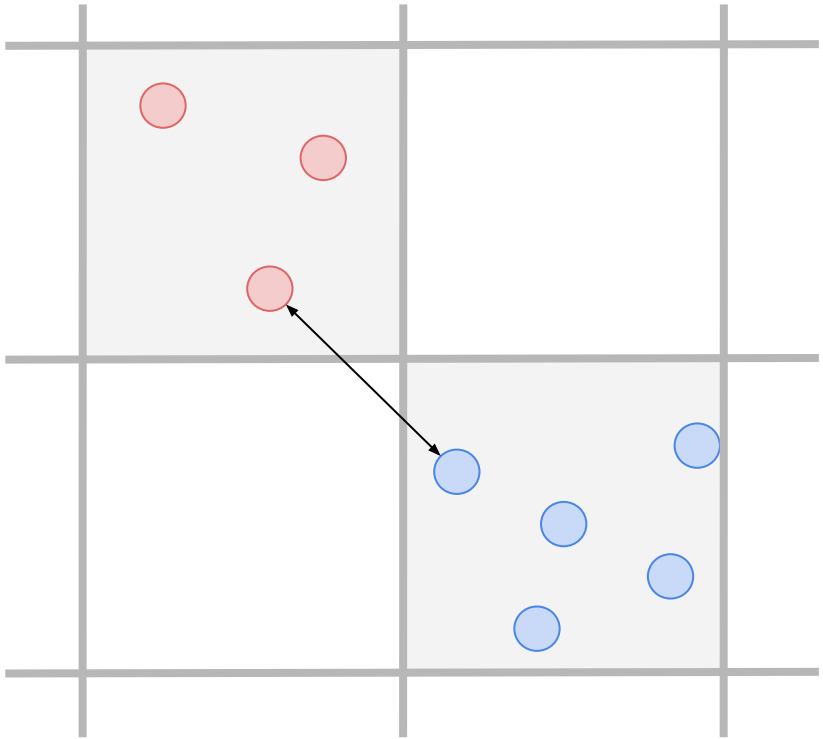


DBSCAN High-level Algorithm (2D)

- Construct cells
- Mark core points
- Cluster core points (cell graph)
- Cluster border points

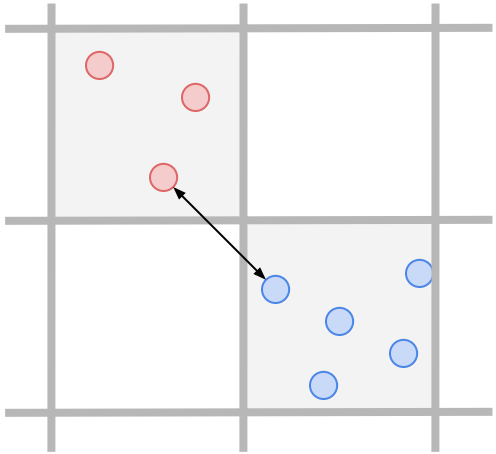
- Cell graph + connected components
- Connect “core cells” (cell with >=1 core points) if they contain core points with <ε distance
- Connected components
  - O(n) expected work
  - O(log n) depth w.h.p.





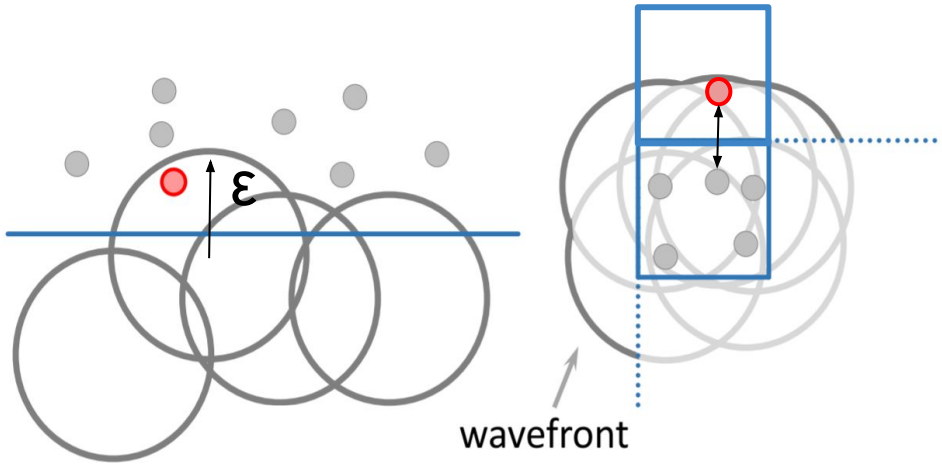
Cell graph - Bichromatic Closest Pair

- Check all pairs of points; pick the minimum
- $O(n^2)$ work
- O(log n) depth
- Practice
  - Simple optimization to get rid of most points



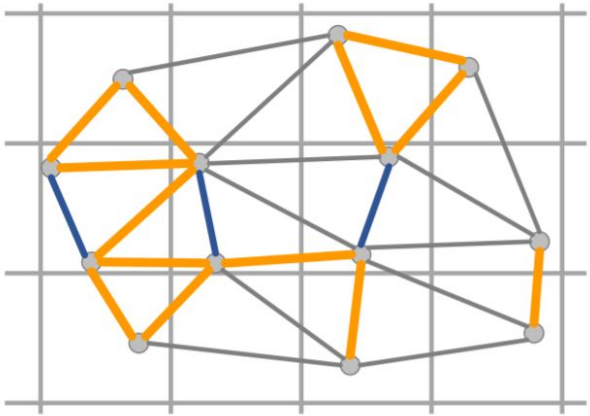
Cell graph - Unit-spherical emptiness checking (USEC)

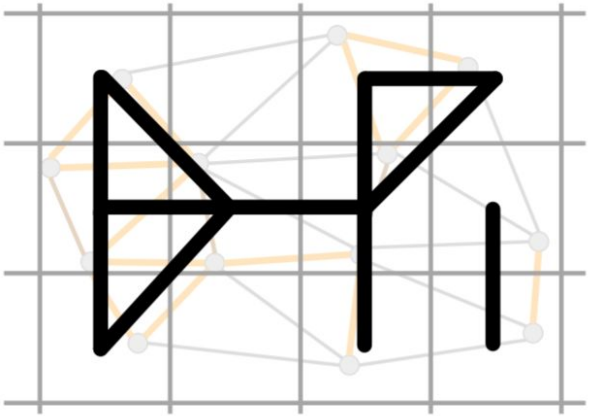
- Serial version, Gan and Tao (2015)
- Create “wavefront” between two cells and check for emptiness
- O(n log n) expected work
- $O(\log^2 n)$ depth w.h.p.
- Practice
  - Sufficient parallelism on grids
  - Serial construction



Cell graph - Delaunay Triangulation

- Serial version, Gan and Tao (2015)
- Connect two cells if they are connected by a Delaunay edge of weight < ε
- Reif & Sen (1992)
  - O(n log n) work
  - O(log n) depth

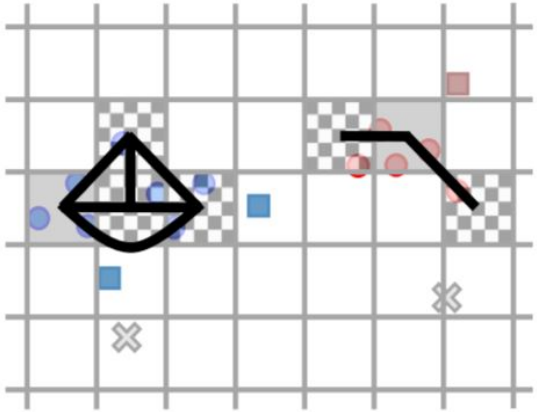




DBSCAN High-level Algorithm (2D)

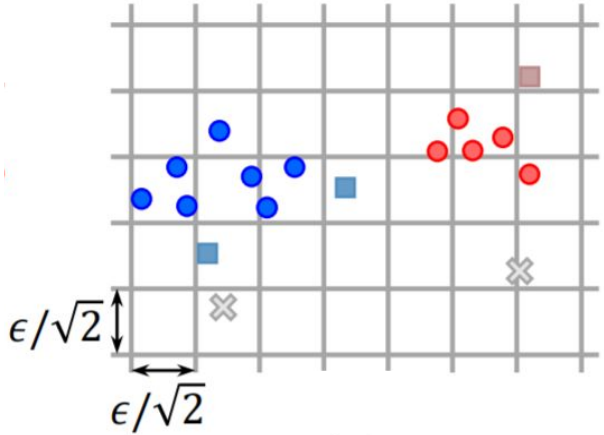
- Construct cells
- Mark core points
- Cluster core points (cell graph)
- Cluster border points

- Assign each border point to close (<ε) core points
- O(n · minPts) work
  O(log n) depth w.h.p.

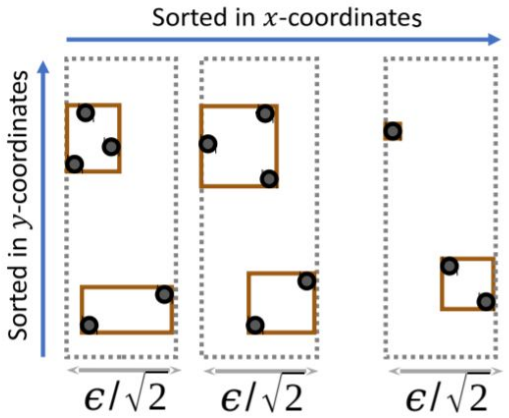


Two types of cells

- Grid cell (Serial version, Gunawan 2013)

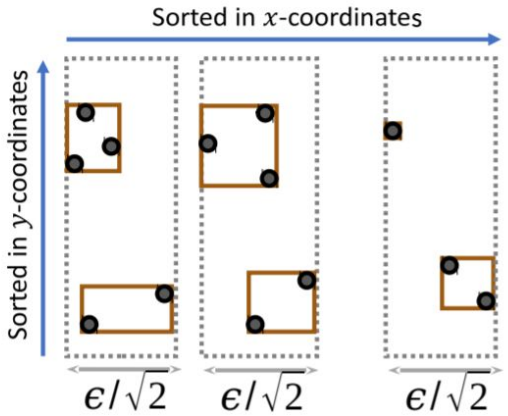


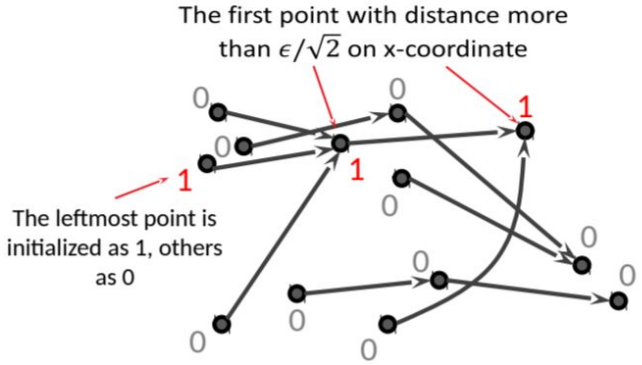
- Box cell (Serial version, Gunawan 2013)



Box cell construction in parallel

- Vertical columns + Horizontal columns
- Binary search to find next boundary
- Pointer jumping to mark column starts
- O(log n) depth

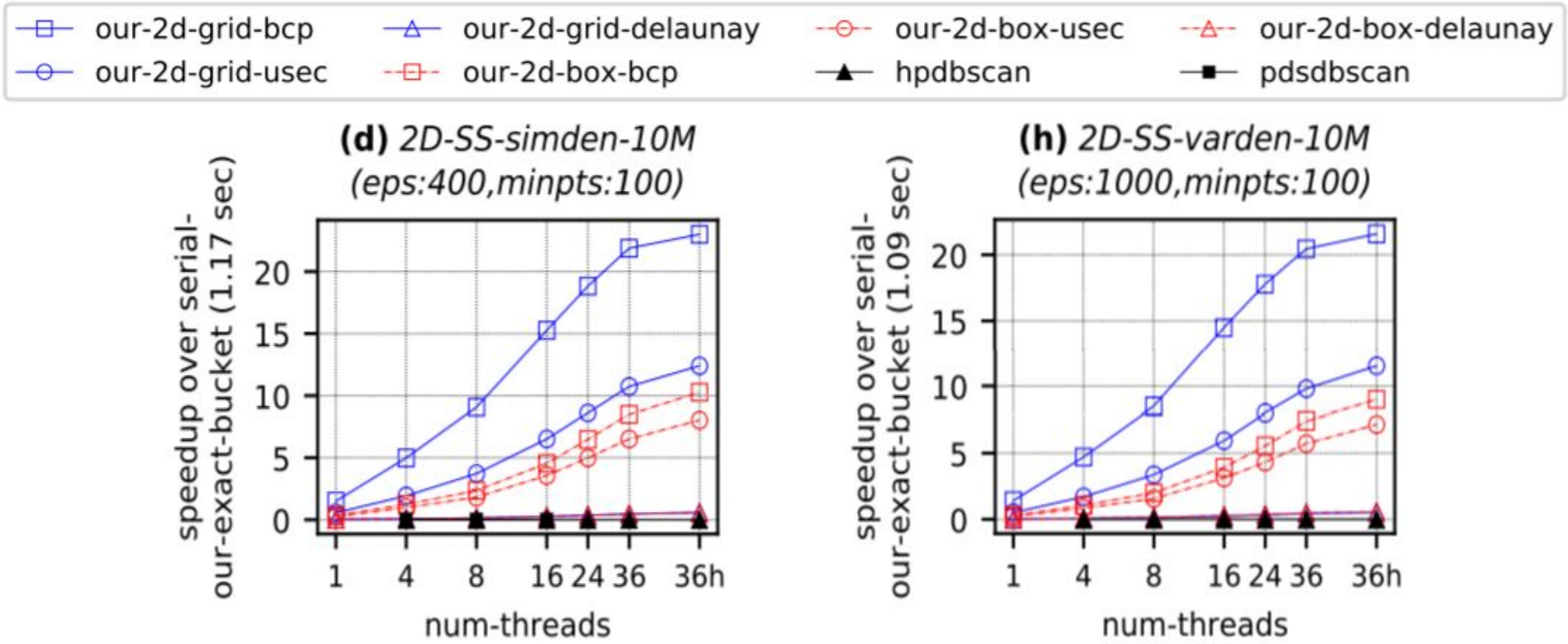




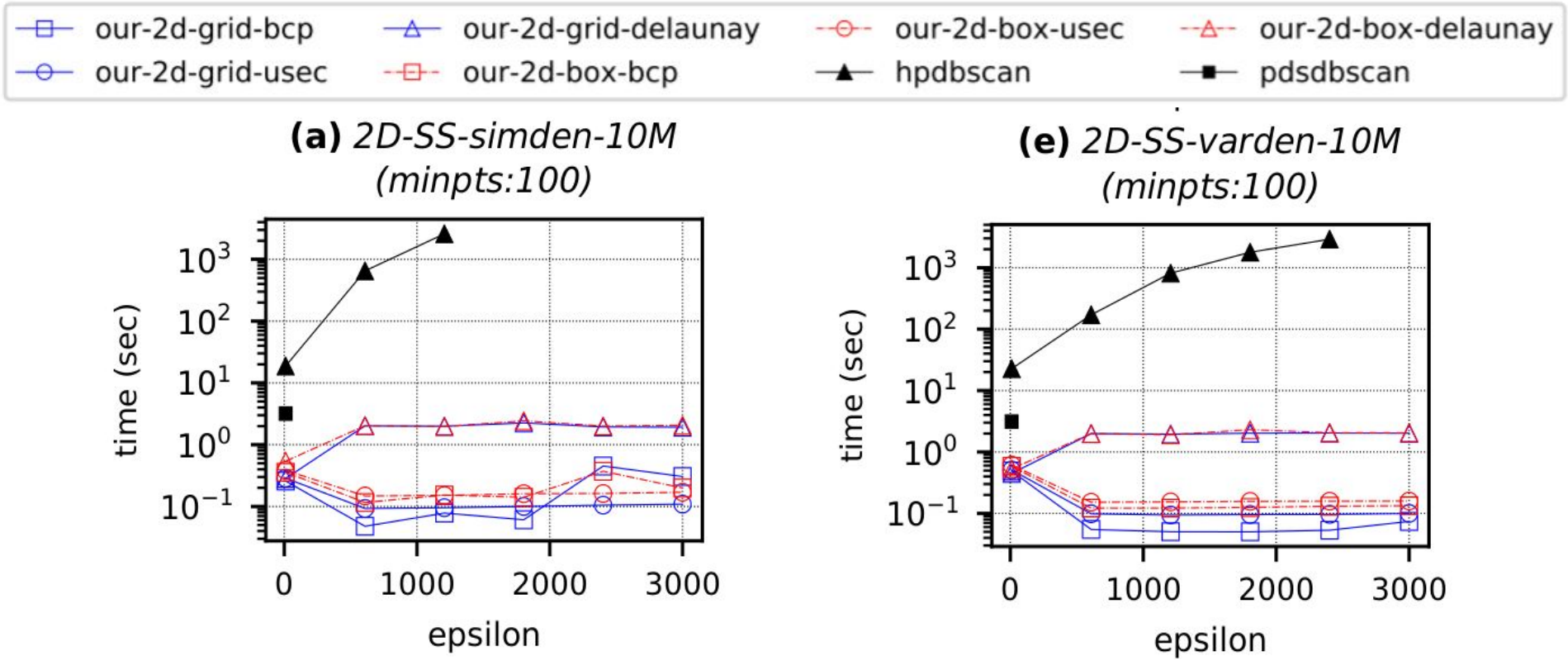
Running Times for 2D

Cell construction	**Grid based**	O(n) expected work; O(log n) depth w.h.p.		
	Box based	O(n log n) work; O(log n) depth		
Mark core		O(n) work; O(log n) depth w.h.p.		
Cell graph		**BCP**	**USEC**	**Delaunay**
		$O(n^2)$ work; O(log n) depth	O(n log n) work; $O(\log^2 n)$ depth	O(n log n) work; O(log n) depth
Connected components		O(n) expected work; O(log n) depth w.h.p.		
Cluster border		O(n) work; O(log n) depth w.h.p		
Overall		$O(n^2)$ expected work; O(log n) depth w.h.p.	O(n log n) expected work; $O(\log^2 n)$ depth w.h.p.	O(n log n) expected work; O(log n) depth w.h.p.

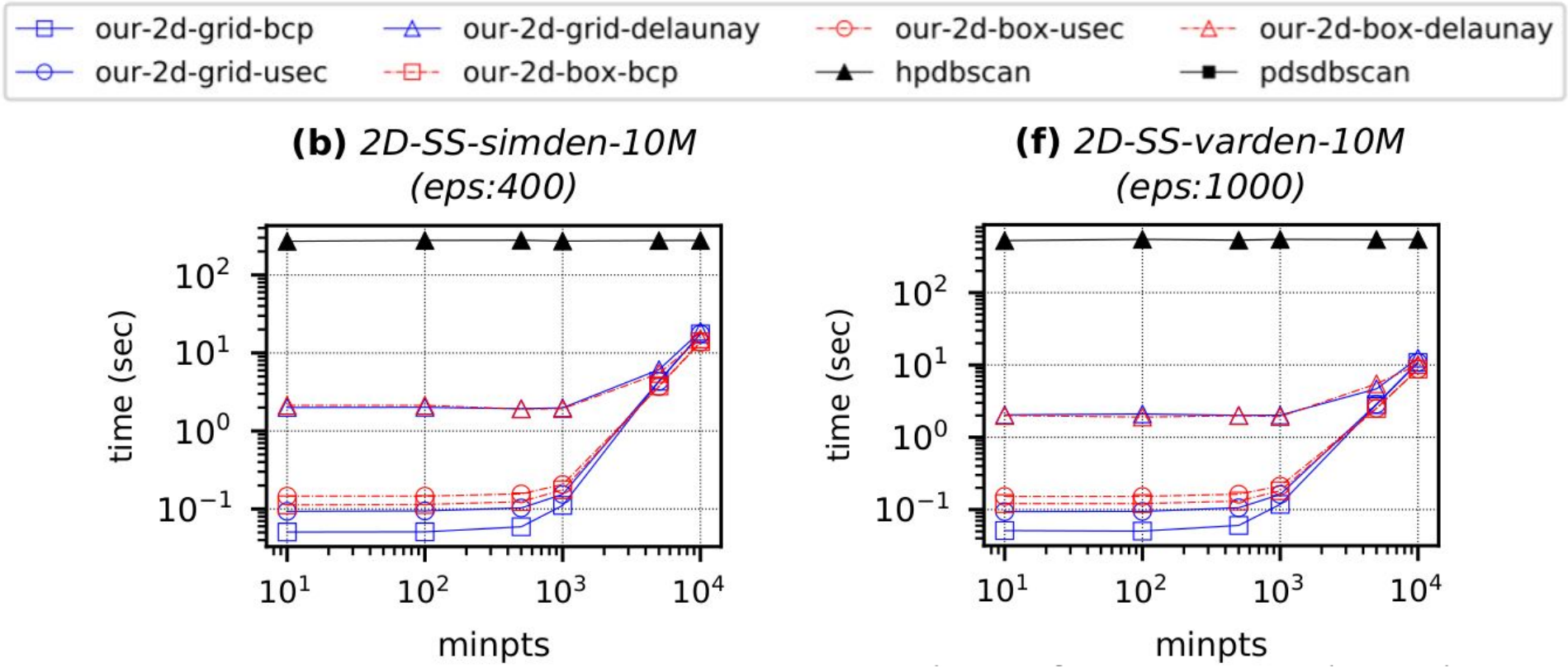
2D Results


our-2d-grid-bcp
our-2d-grid-delaunay
our-2d-box-usec
our-2d-box-delaunay
our-2d-grid-usec
our-2d-box-bcp
hpdbscan
pdsdbscan
(d) 2D-SS-simden-10M
(eps:400,minpts:100)
speedup over serial-our-exact-bucket (1.17 sec)
num-threads
(h) 2D-SS-varden-10M
(eps:1000,minpts:100)
speedup over serial-our-exact-bucket (1.09 sec)
num-threads

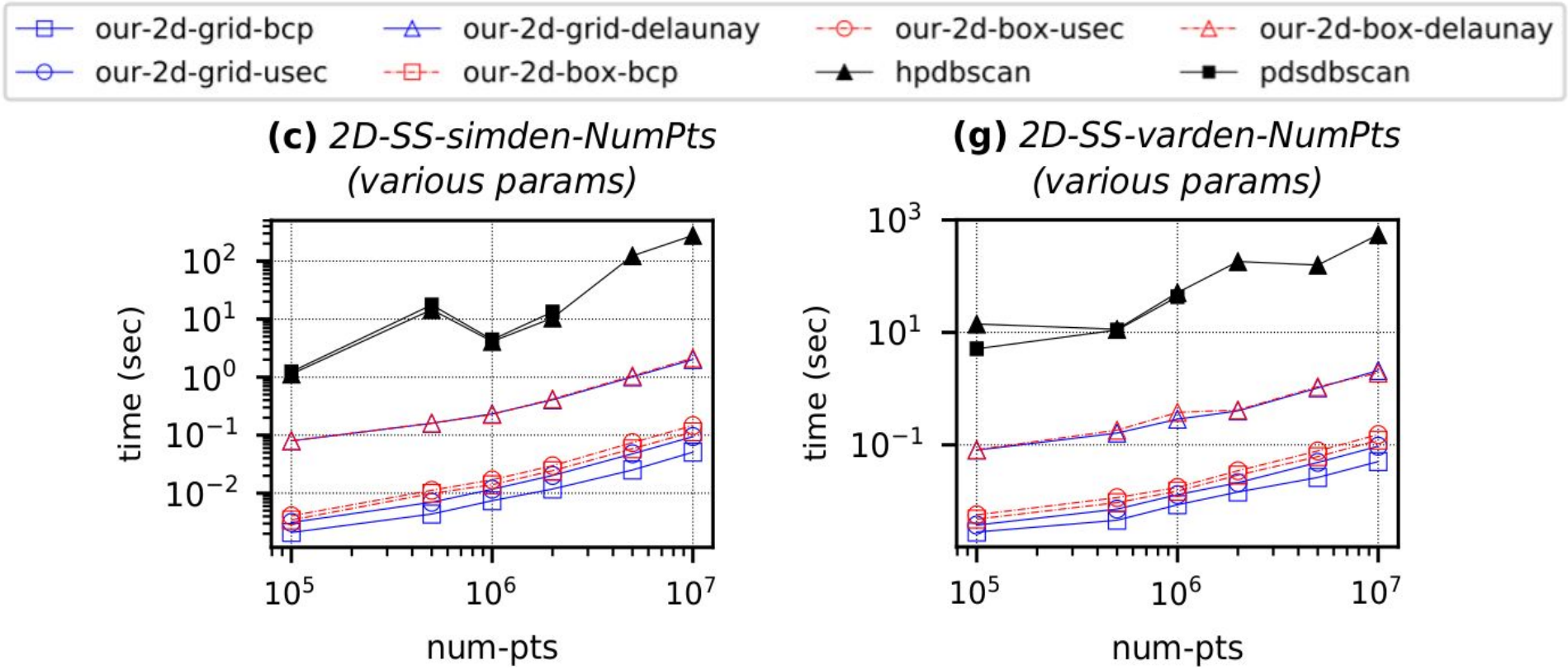
2D Results


our-2d-grid-bcp
our-2d-grid-delaunay
our-2d-box-usec
our-2d-box-delaunay
our-2d-grid-usec
our-2d-box-bcp
hpdbscan
pdsdbscan
(a) 2D-SS-simden-10M
(minpts:100)
time (sec)
epsilon
(e) 2D-SS-varden-10M
(minpts:100)
time (sec)
epsilon

2D Results

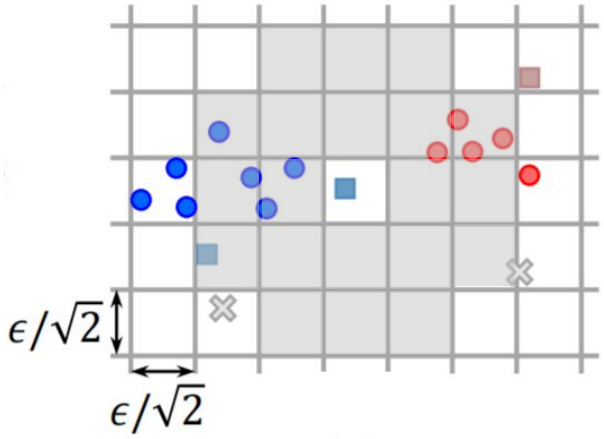

our-2d-grid-bcp
our-2d-grid-delaunay
our-2d-box-usec
our-2d-box-delaunay
our-2d-grid-usec
our-2d-box-bcp
hpdbscan
pdsdbscan
(b) 2D-SS-simden-10M
(eps:400)
(f) 2D-SS-varden-10M
(eps:1000)
time (sec)
minpts
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}

2D Results


our-2d-grid-bcp
our-2d-grid-delaunay
our-2d-box-usec
our-2d-box-delaunay
our-2d-grid-usec
our-2d-box-bcp
hpdbscan
pdsdbscan
(c) 2D-SS-simden-NumPts
(various params)
time (sec)
num-pts
(g) 2D-SS-varden-NumPts
(various params)
time (sec)
num-pts

Higher dimensional DBSCAN

- More complicated cell-neighbor finding
- Parallel kd-tree for answer spatial range queries



21 neighbor grids

$\approx 5^d$

Optimization 1 - UnionFind

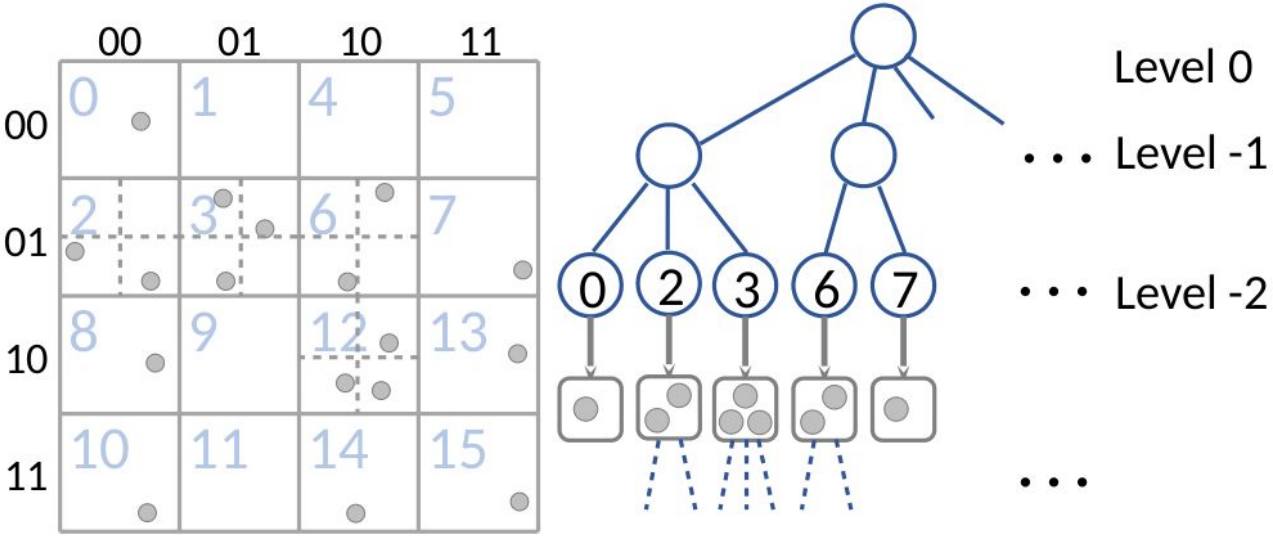
- Serial version, Gan and Tao (2015)
- BCP & USEC
- Compute CC while constructing the cell graph
- Prunes independent cell-cell connectivity queries

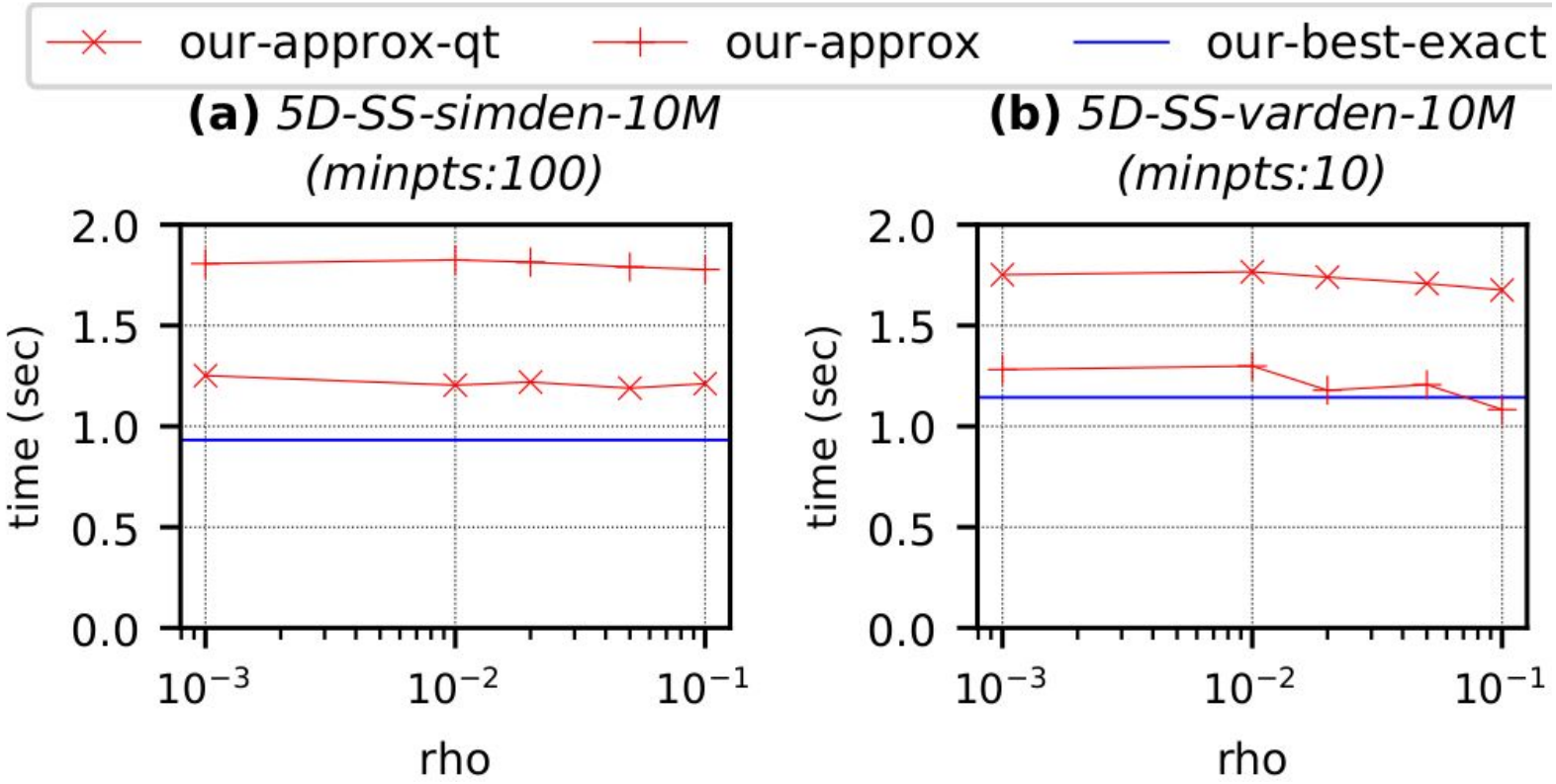
Optimization 2 - Bucketing

- Cells with more points -> more likely connected
  - Connect them early
  - More pruning
- Serial: sort + iterate
- Parallel: sort + bucketing

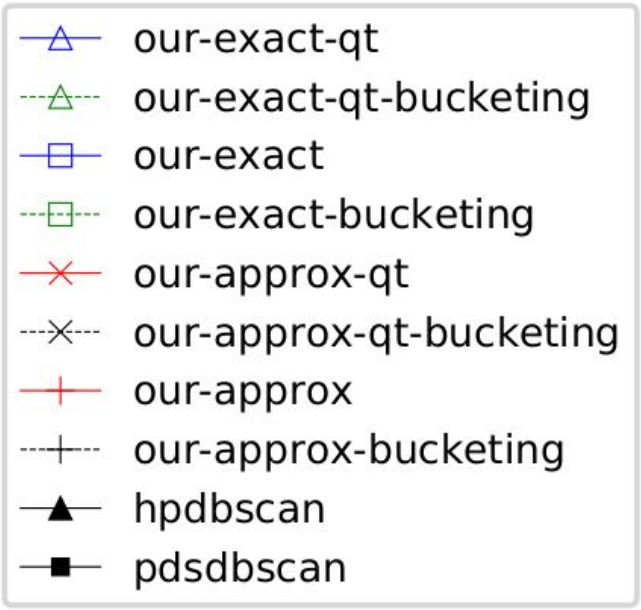
Cell connectivity using spatial quadtree

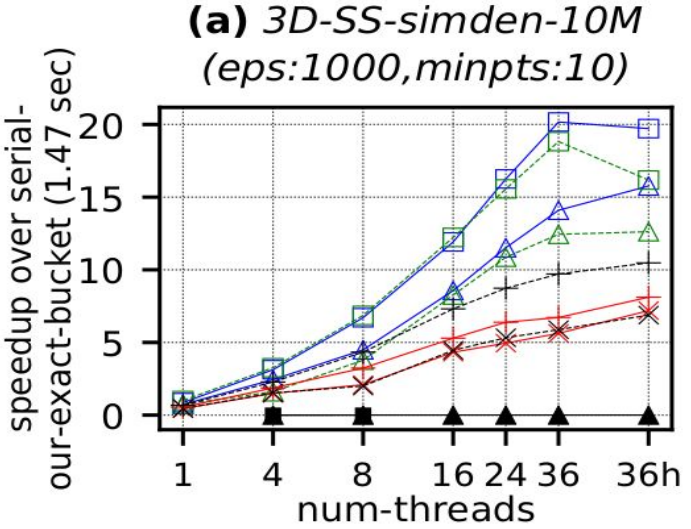
- For both exact DBSCAN and approximate DBSCAN

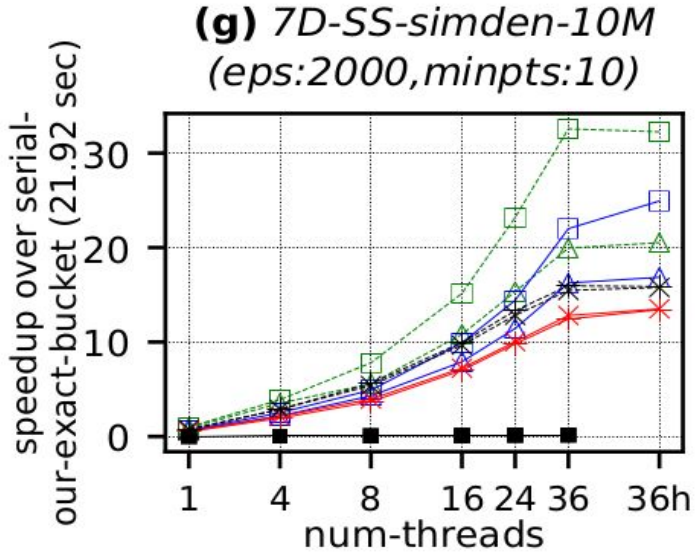



our-approx-qt
our-approx
our-best-exact
(a) 5D-SS-simden-10M
(minpts:100)
(b) 5D-SS-varden-10M
(minpts:10)
time (sec)
2.0
1.5
1.0
0.5
0.0
10^{-3}
10^{-2}
10^{-1}
rho

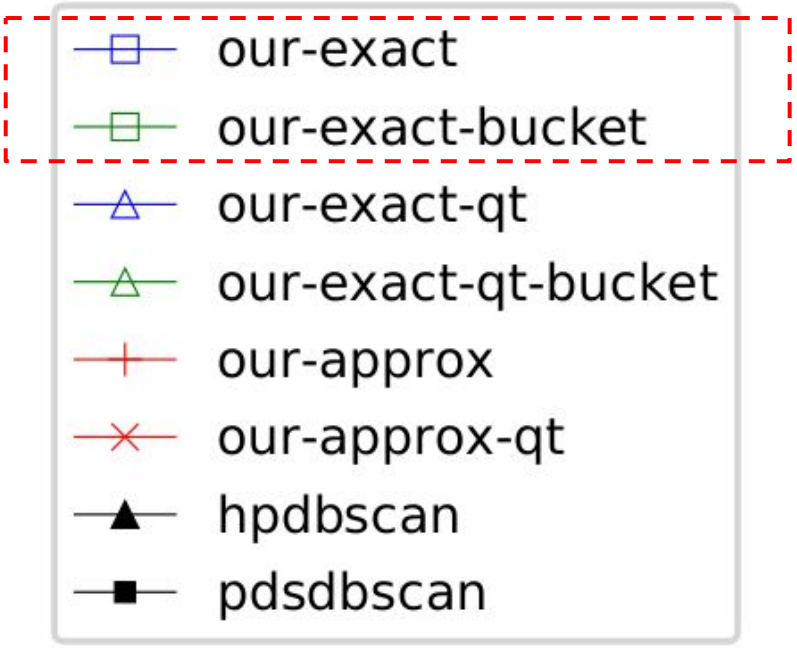
Experiments

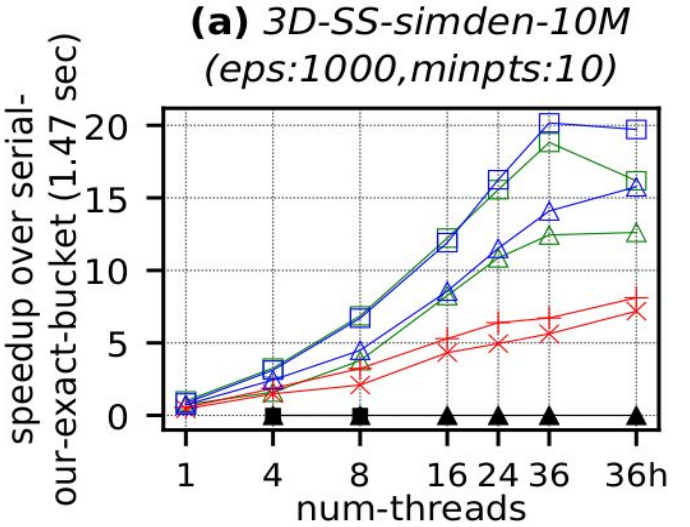


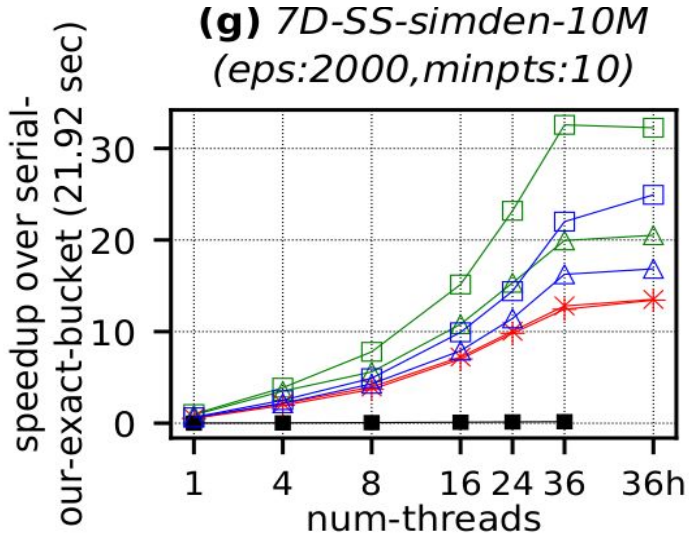




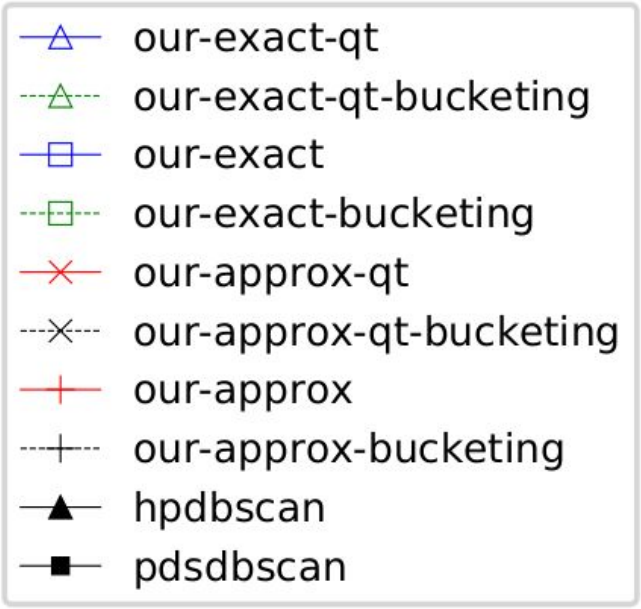
Experiments

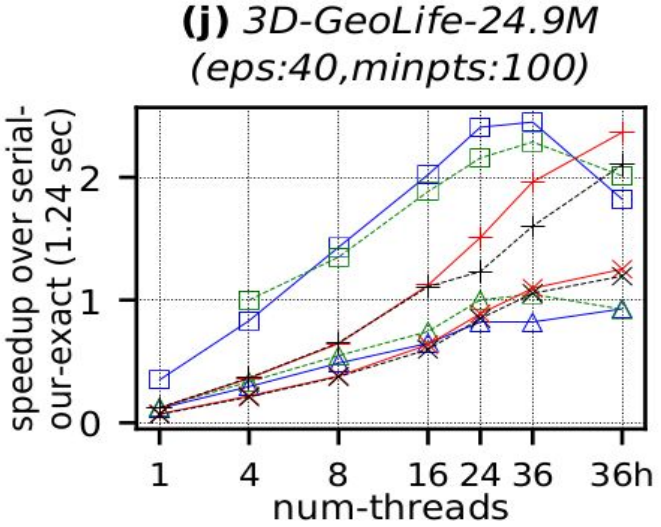


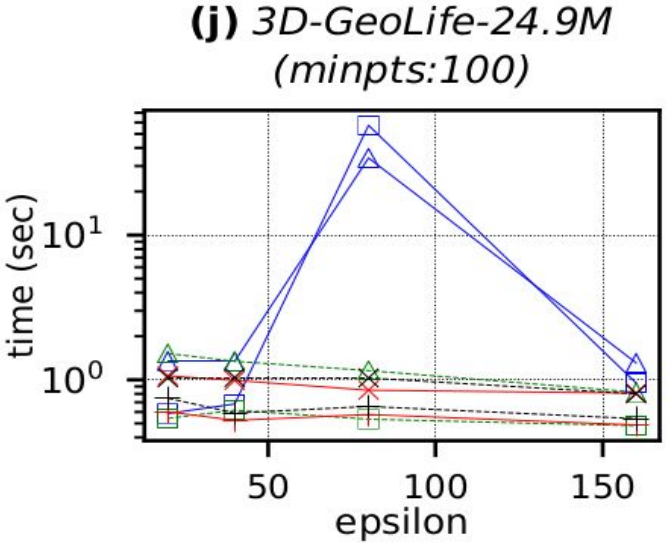




Experiments

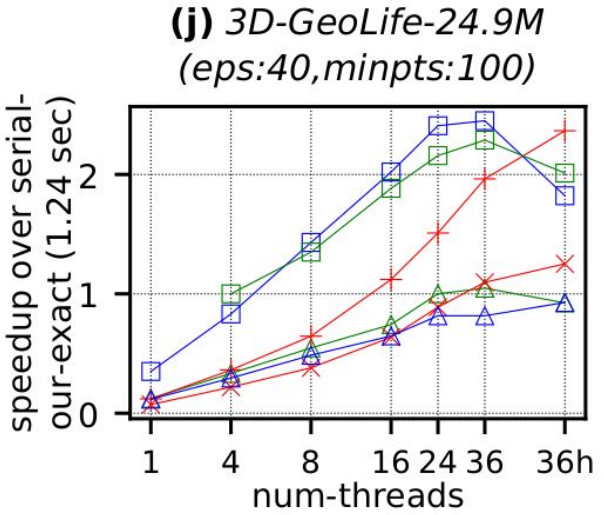


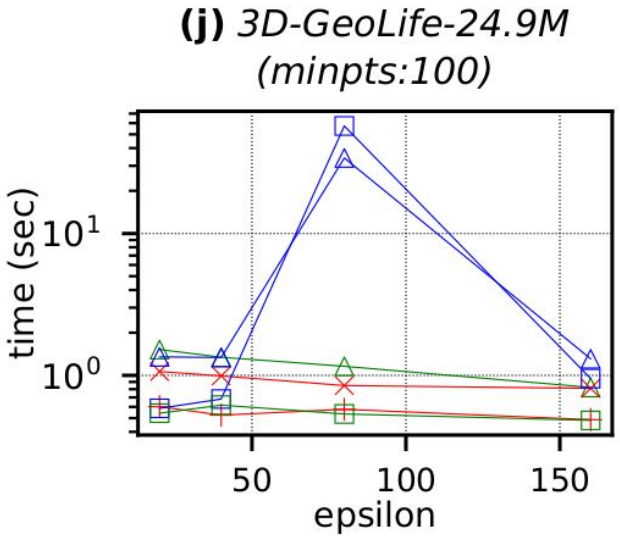




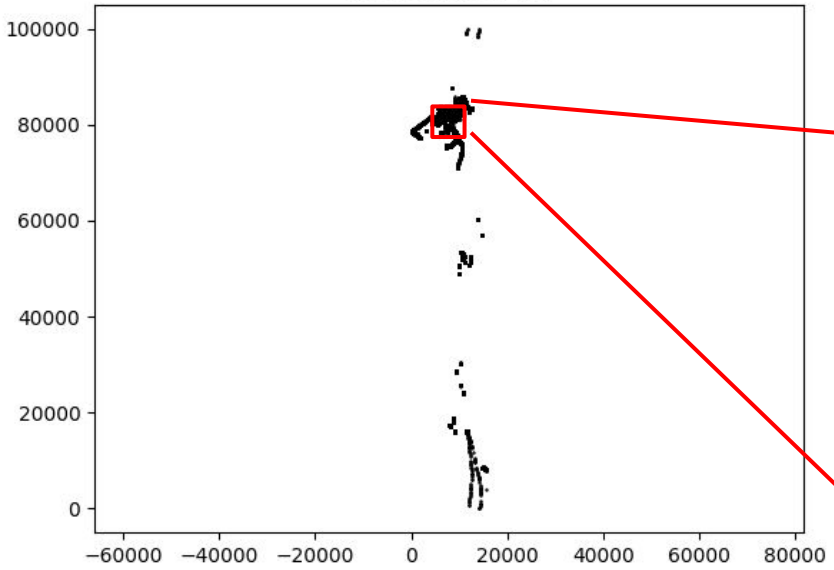
Experiments

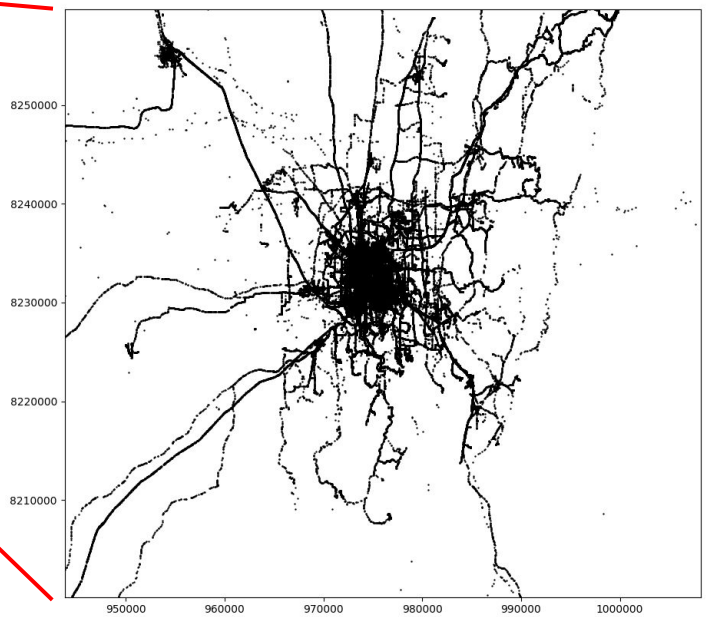
- our-exact
- our-exact-bucket
- our-exact-qt
- our-exact-qt-bucket
- our-approx
- our-approx-qt
- hpdbscan
- pdsdbscan





GeoLife


100000
80000
60000
40000
20000
0
−60000
−40000
−20000
0
20000
40000
60000
80000


8250000
8240000
8230000
8220000
8210000
950000
960000
970000
980000
990000
1000000

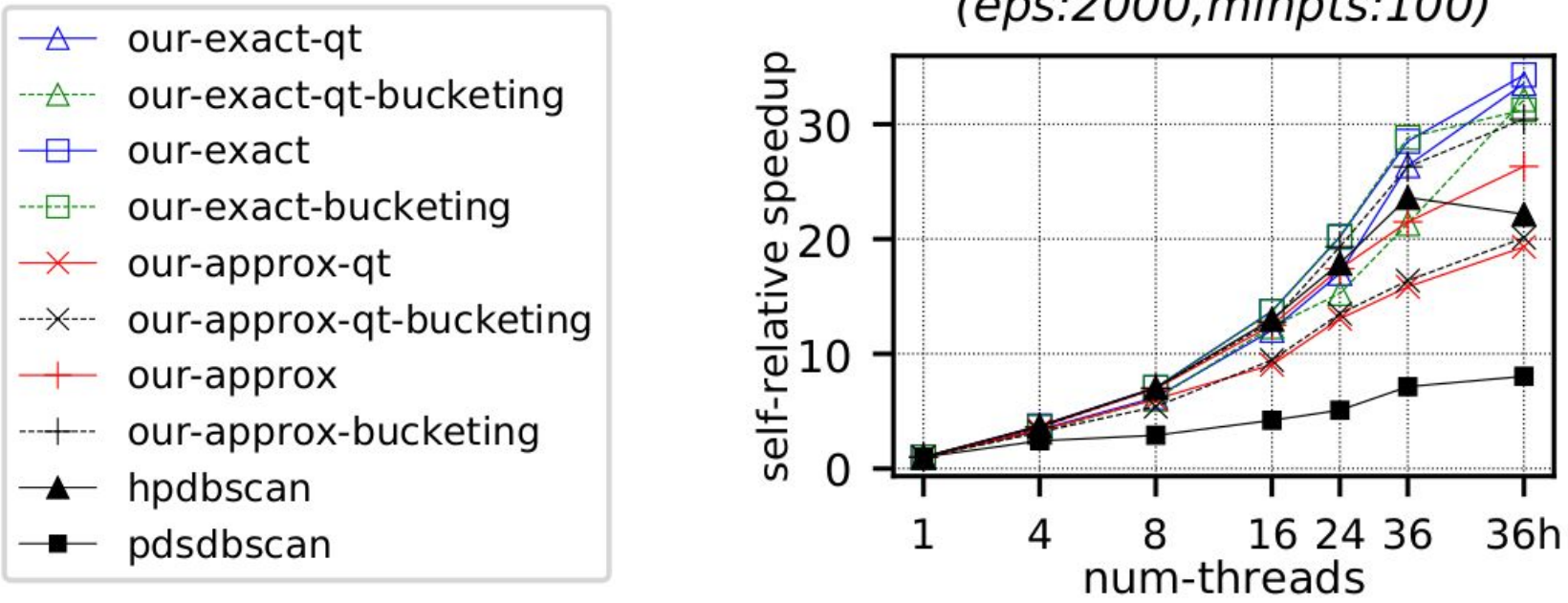
Experiments

	#data	*dimension*
GeoLife	*24.9M*	*3*
Cosmo50	*321M*	*3*
OpenStreetMap	*2770M*	*2*
TeraClickLog	*4373M*	*13*

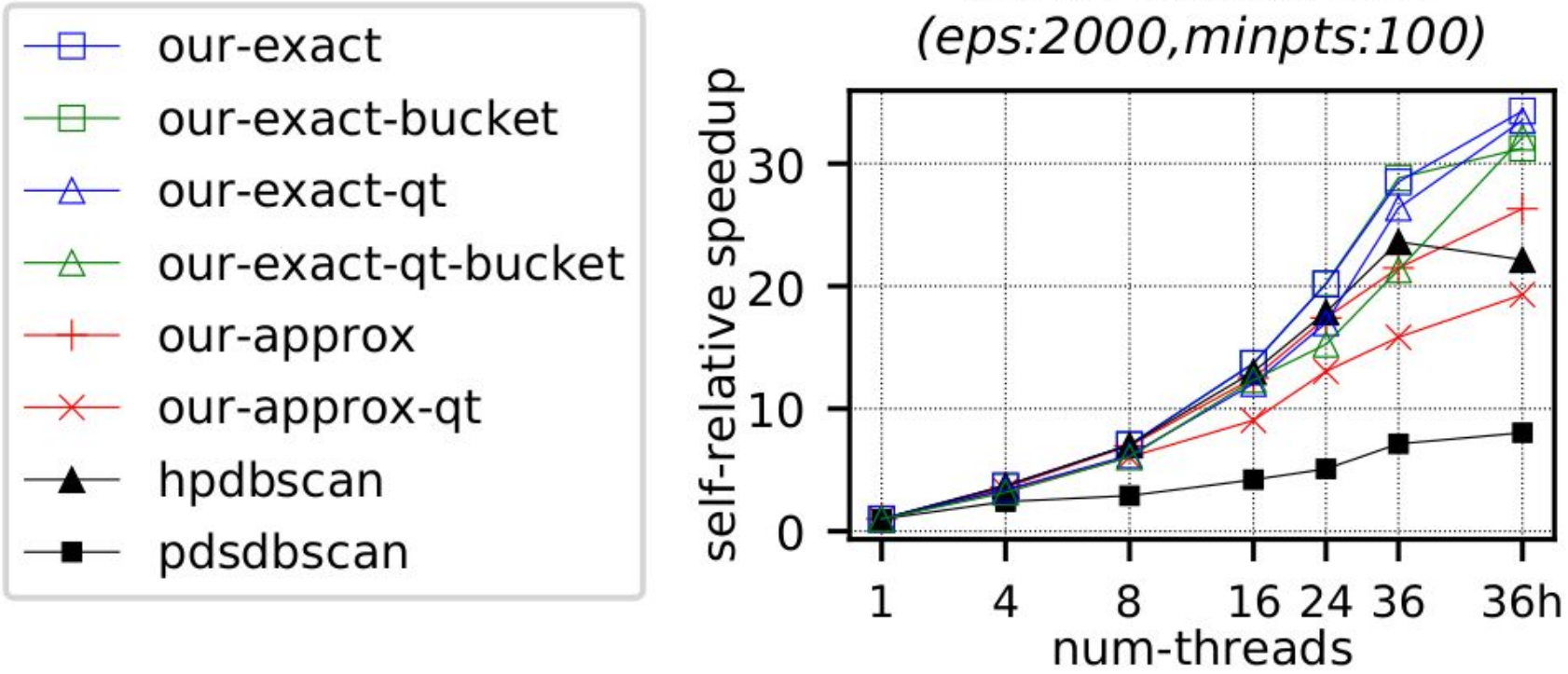
	GeoLife				Cosmo50			
ϵ	20	40	80	160	0.01	0.02	0.04	0.08
our-exact	0.541	0.617	0.535	0.482	41.8	5.51	4.69	3.03
rpdbscan (our machine)	29.13	27.92	32.04	27.81	3750	562.0	576.9	672.6
rpdbscan ([74])	36	33	28	27	960	504	438	432

	OpenStreetMap				TeraClickLog			
ϵ	0.01	0.02	0.04	0.08	1500	3000	6000	12000
our-exact	41.4	43.2	40	44.5	26.8	26.9	27.0	27.6
rpdbscan (our machine)	–	–	–	–	–	–	–	–
rpdbscan ([74])	3000	1720	1200	840	15480	7200	3540	1680

Experiments - Self-relative speedups



Experiments - Self-relative speedups



Conclusion

- Practical parallel algorithms for 2D exact DBSCAN, and higher dimensional exact and approximate DBSCAN with work bounds matching the best sequential algorithms, and polylogarithmic depth.
- Highly-optimized implementations.
- A comprehensive experimental evaluation
  - 2-38x self-relative speedup (36 cores)
  - 5-33x speedup over best sequential (36 cores)
  - 14-6102x faster than existing parallel implementations
  - Processes largest dataset known for DBSCAN (> 4 billion points) on just one machine

Future work

- Our code will be available online soon

Questions