Scalable Processing of Moving Flock Patterns

Andres Calderon · acald013@ucr.edu

University of California, Riverside

November 4, 2024

Outline

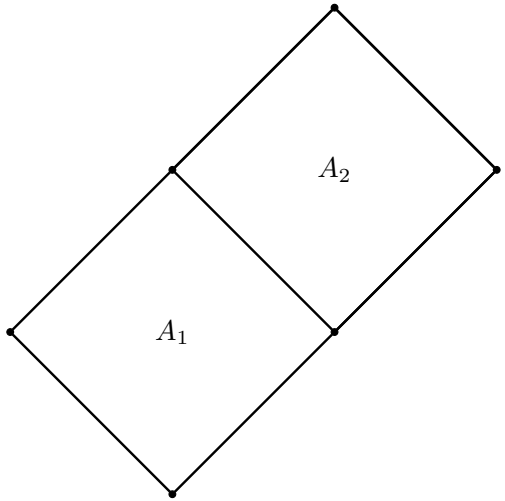
- Scalable overlay operations over DCEL polygons layers
- An Extension On Scalable DCEL Overlay Operations
- Scalable Processing of Moving Flock Patterns

Outline

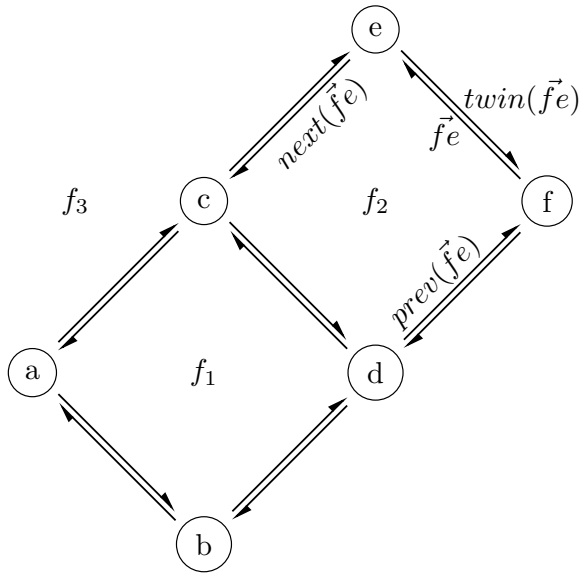
- **Scalable overlay operations over DCEL polygons layers**
- An Extension On Scalable DCEL Overlay Operations
- Scalable Processing of Moving Flock Patterns

What is a DCEL?

- **Doubly Connected Edge List** - DCEL.
- A spatial data structure to **represent planar subdivisions** of surfaces.
- Represent topological and geometric information as vertices, edges, and faces.
- **Applications**: polygon overlays, polygon triangulation and their applications in surveillance, robot motion planing, circuit board printing, etc.

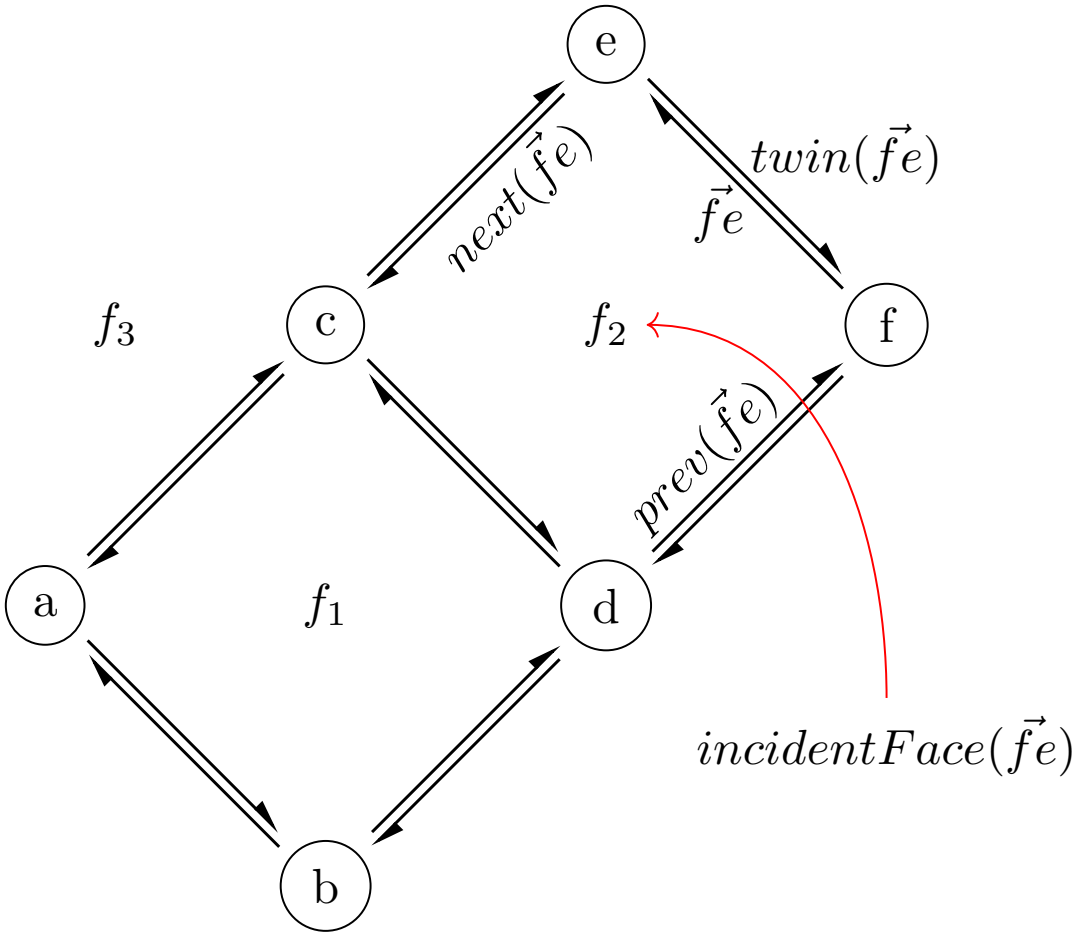


Planar subdivision



DCEL representation

- DCEL uses three tables: Vertices, Faces and Half-edges.



DCEL description

- DCEL uses three tables: Vertices, Faces and Half-edges.

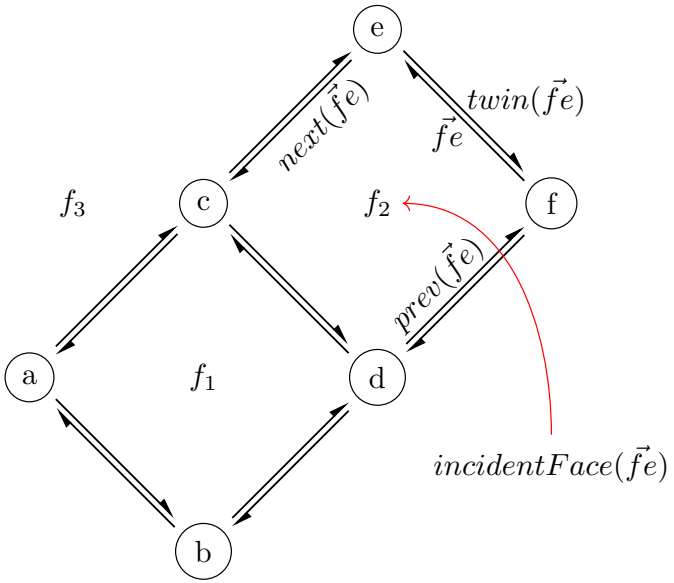


Table 1: **Vertex records.**

vertex	coordinates	incident edge
a	(0,2)	$\vec{ba}$
b	(2,0)	$\vec{db}$
⋮	⋮	⋮
e	(4,6)	$\vec{fe}$
f	(6,4)	$\vec{df}$
⋮	⋮	⋮

Table 2: **Face records.**

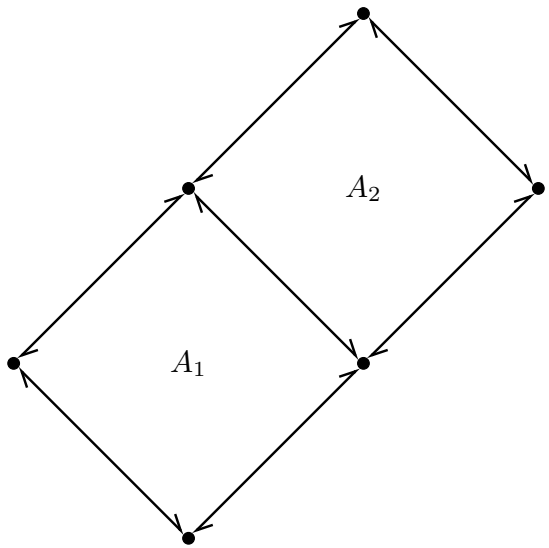
face	boundary edge	hole list
f_1	$\vec{ab}$	*nil*
f_2	$\vec{fe}$	*nil*
f_3	*nil*	*nil*

Table 3: **Half-edge records.**

half-edge	origin	face	twin	next	prev
$\vec{fe}$	f	f_2	$\vec{ef}$	$\vec{ec}$	$\vec{df}$
$\vec{ca}$	c	f_1	$\vec{ac}$	$\vec{ab}$	$\vec{dc}$
$\vec{db}$	d	f_3	$\vec{bd}$	$\vec{ba}$	$\vec{fd}$
⋮	⋮	⋮	⋮	⋮	⋮

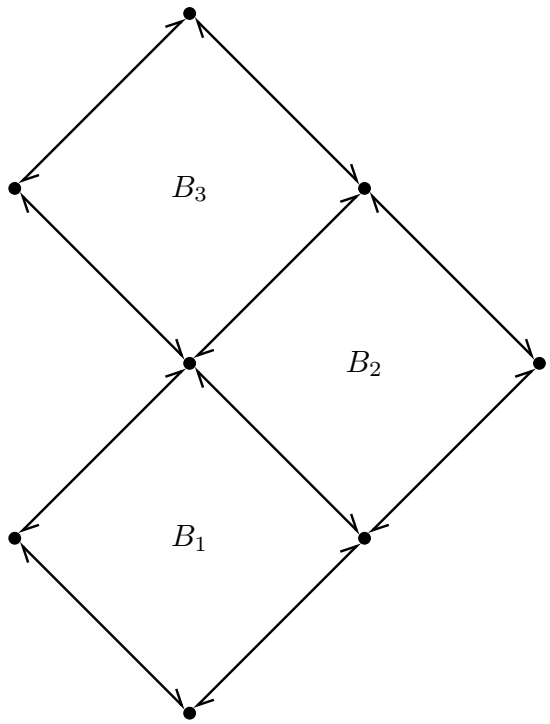
DCEL Advantages

- **Efficiency**: very efficient for computation of *overlay operators*.
- **Re-usability**: allows multiple operations over the same DCEL.
- **Pipelining**: the output of a DCEL operator can be input to another DCEL operator.



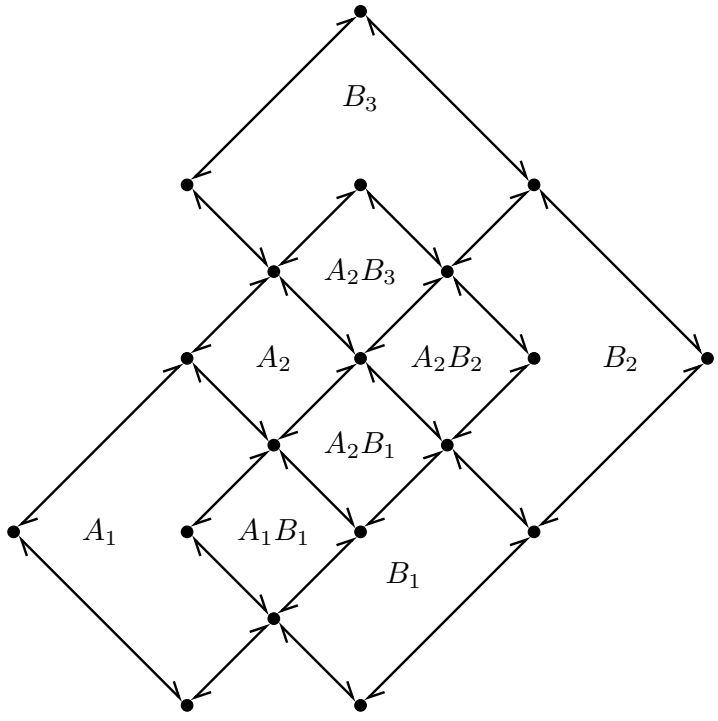
DCEL Advantages

- **Efficiency**: very efficient for computation of *overlay operators*.
- **Re-usability**: allows multiple operations over the same DCEL.
- **Pipelining**: the output of a DCEL operator can be input to another DCEL operator.



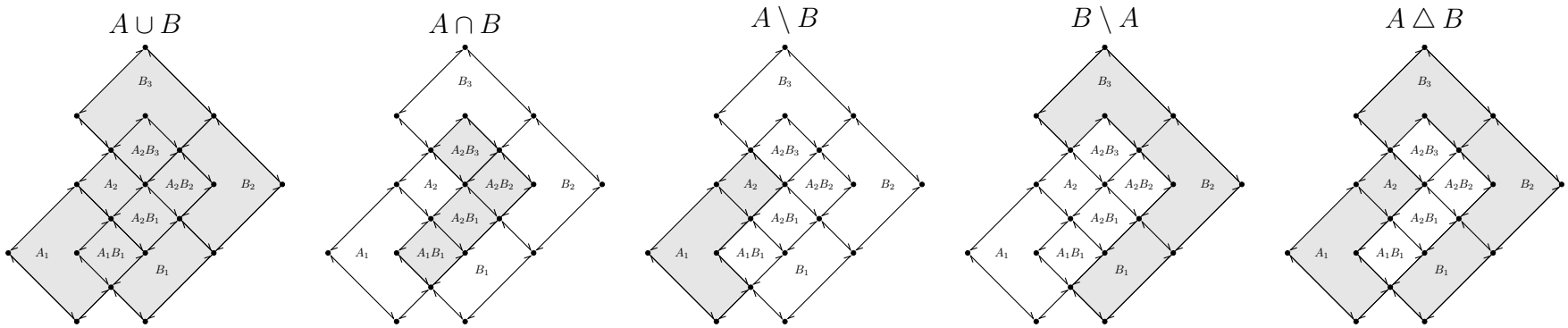
DCEL Advantages

- **Efficiency**: very efficient for computation of *overlay operators*.
- **Re-usability**: allows multiple operations over the same DCEL.
- **Pipelining**: the output of a DCEL operator can be input to another DCEL operator.



DCEL Advantages

- **Efficiency**: very efficient for computation of *overlay operators*.
- **Re-usability**: allows multiple operations over the same DCEL.
- **Pipelining**: the output of a DCEL operator can be input to another DCEL operator.

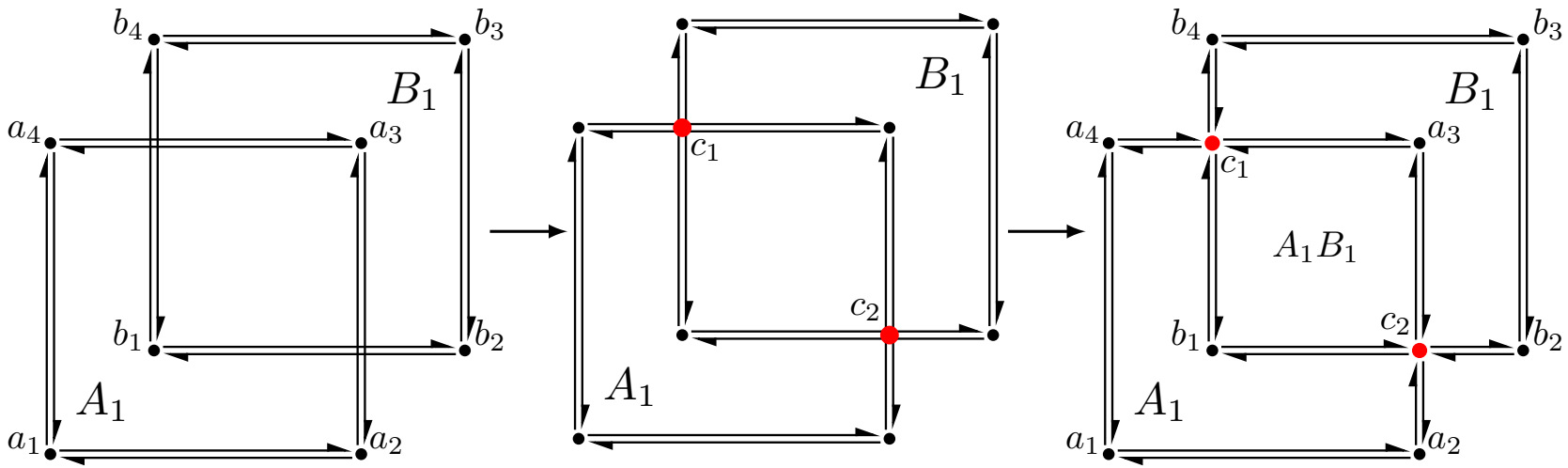


Challenges and Contributions

- Currently only sequential DCEL implementations exist.
- Unable to deal with large datasets (i.e. US Census tracks at national level).
- We propose a *scalable distributed* approach to compute the overlay of two polygon layers using DCELs.
- Distribution enables scalability, but introduces challenges: the ***orphan-cell*** problem and the ***orphan-hole*** problem.

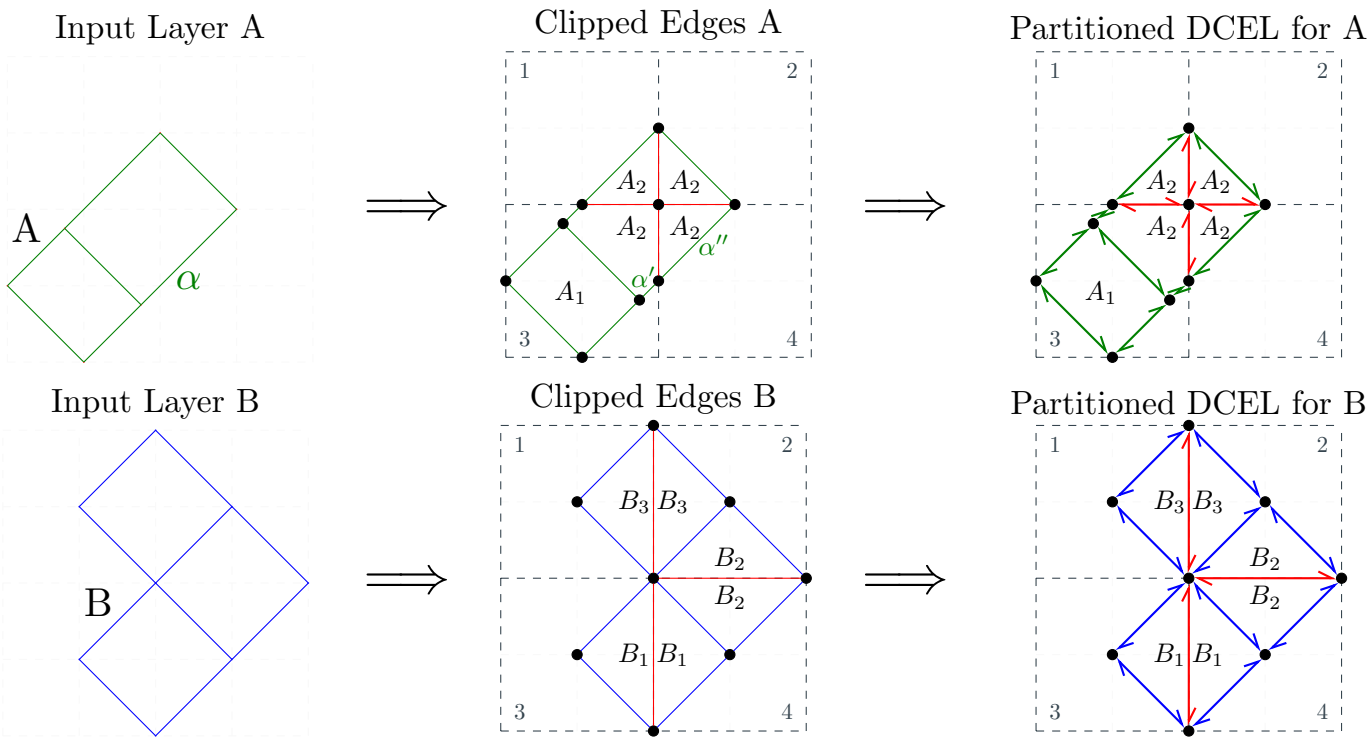
Sequential implementation

- Consider two (simple) input DCELs A_1 and B_1. The sequential algorithm first finds the intersections of half-edges.
- Then, new vertices (e.g. c_1, c_2) are created, half-edges are updated, new faces are added and labeled (e.g. A_1B_1).



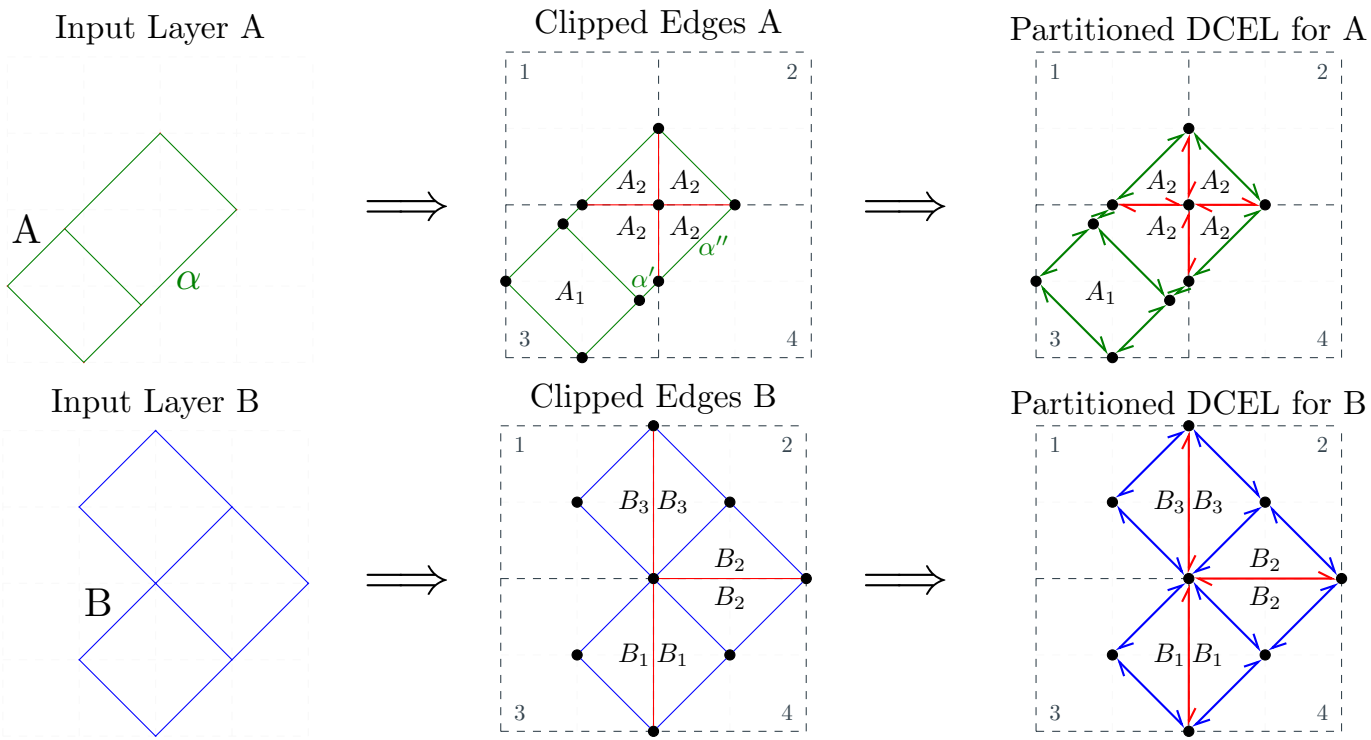
Scalable Implementation

- Two phases: (1) Distributed DCEL construction, and (2) Distributed overlay evaluation.
- Distribution is based on a spatial index (e.g. quadtree)
- Each input DCEL layer (e.g. A, B) is partitioned using the same index

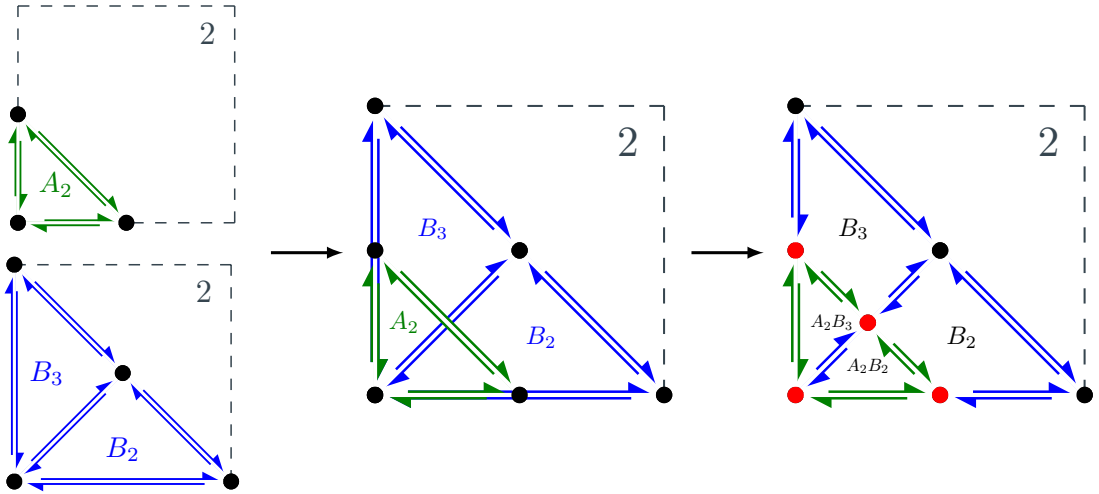


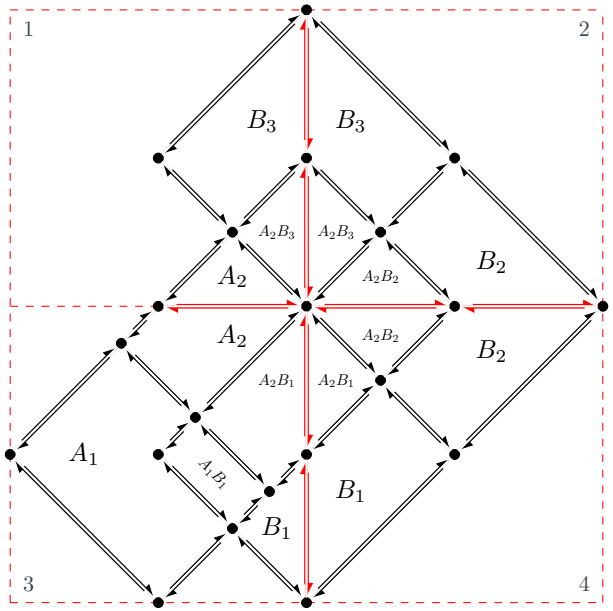
Scalable Implementation

- Each index cell should contain all information needed so that it can compute the overlay DCEL locally
- For each cell to be independent, we need to create "artificial" edges and vertices



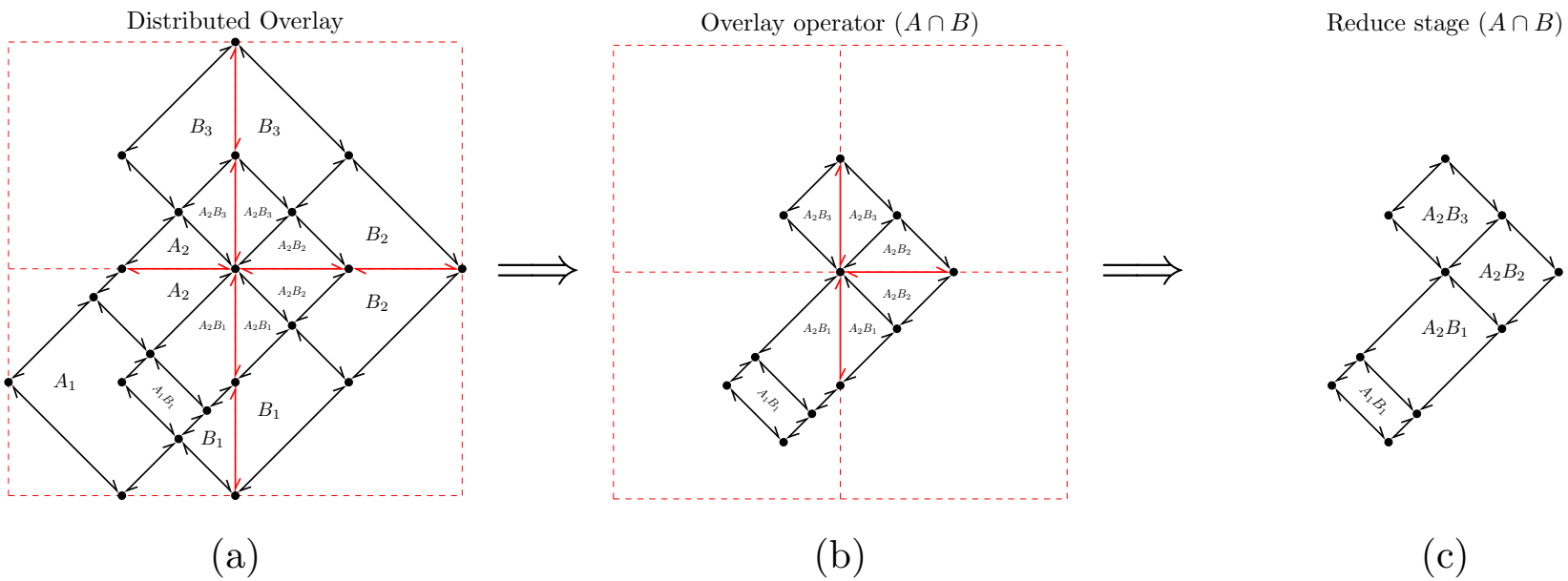
Distributed DCEL construction





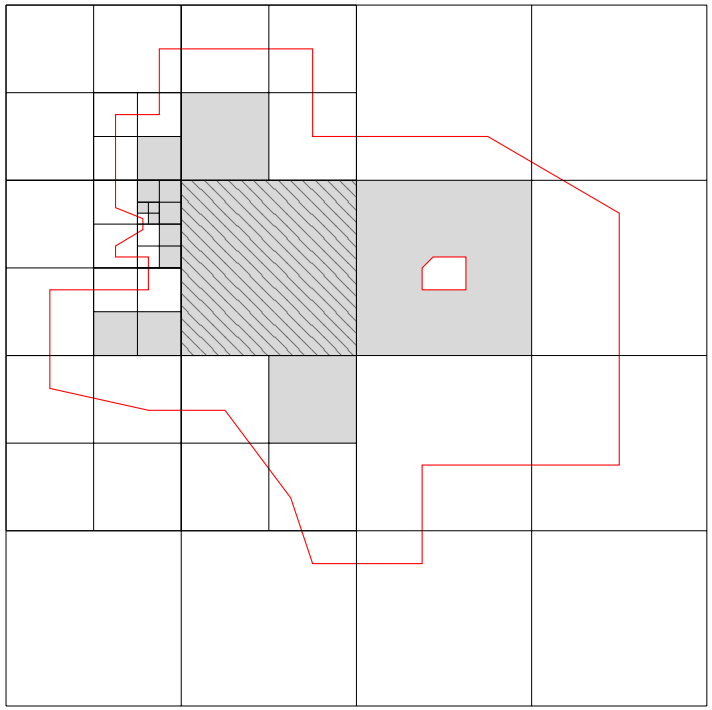
Overlay evaluation

- Answering global overlay queries...
  - To compute a particular overlay operator, we query local DCELs.
  - This work is done independently at each cell (node).
  - SDCEL then collects back all local DCEL answers and computes the final answer (by removing artificial edges and concatenating the resulting faces).



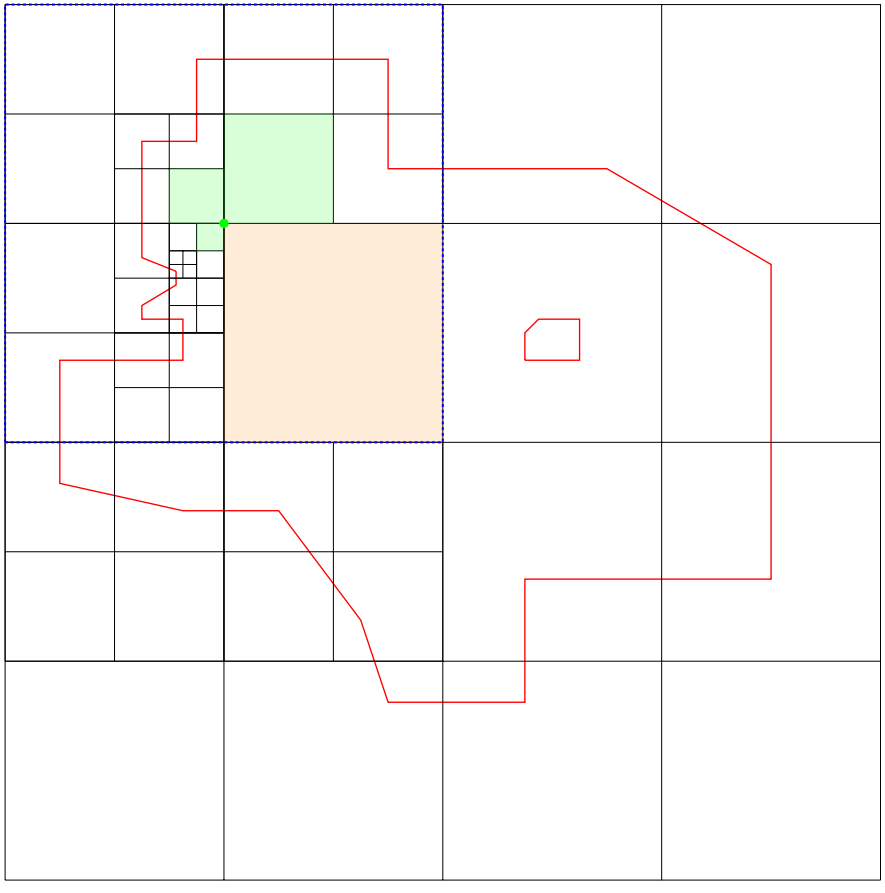
Labeling orphan cells and orphan holes

- We next discuss the **orphan cell** problem (orphan holes are handled similarly).
- A large face (e.g. the red polygon in the figure) can contain cells that do not intersect with any of the face's boundary edges (called *regular edges*).
- Such cells do not contain any label and thus we do not know which face they belong to.



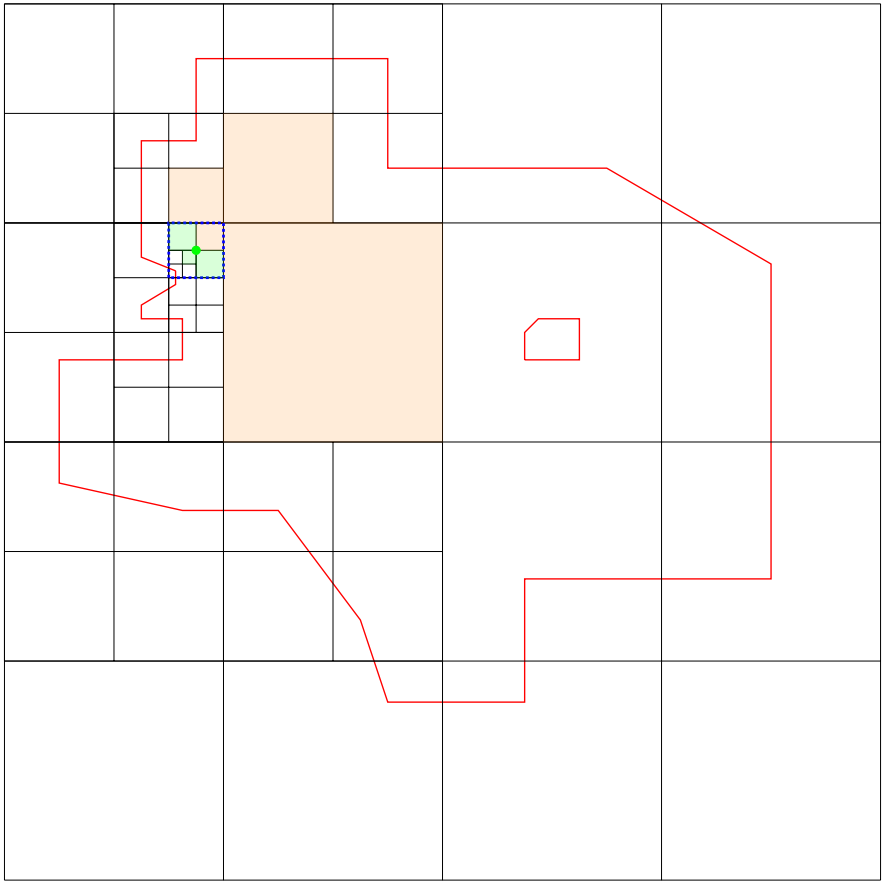
Labeling orphan cells and orphan holes

- We provide an algorithm to efficiently solve the orphan cell problem.



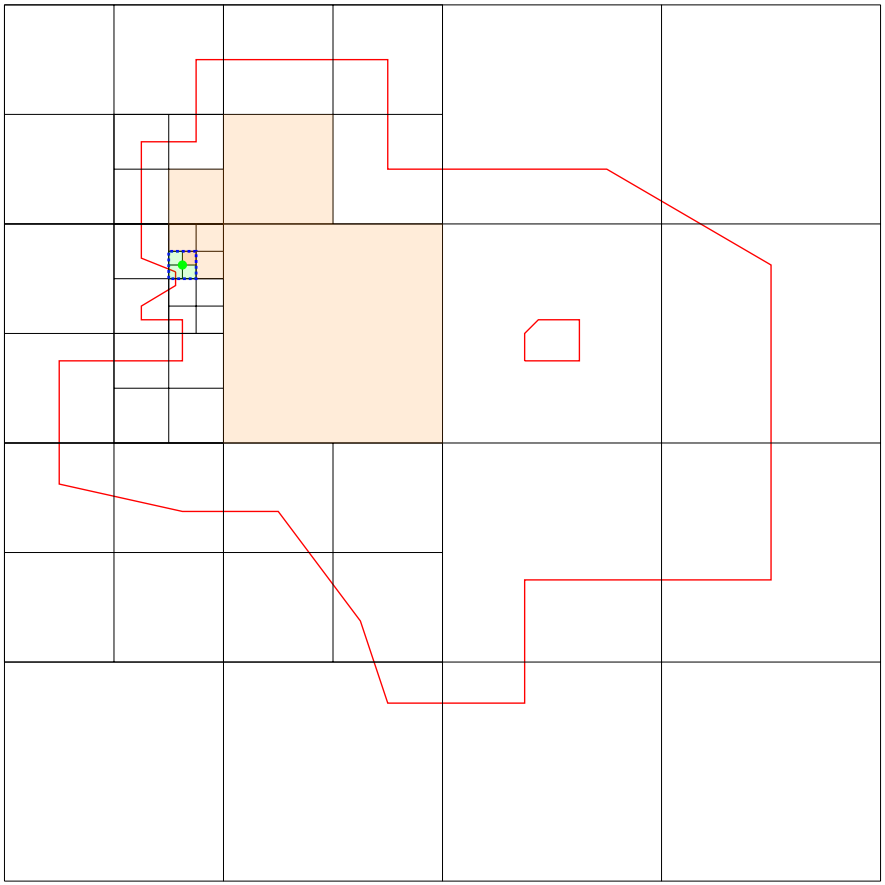
Labeling orphan cells and orphan holes

- We provide an algorithm to efficiently solve the orphan cell problem.



Labeling orphan cells and orphan holes

- We provide an algorithm to efficiently solve the orphan cell problem.



Algorithm 1: getNextCellWithEdges algorithm

```
   Input: a quadtree Q and a list of cells M.
 1 function getNextCellWithEdges(Q,M):
 2     C ← orphan cells in M
 3     foreach orphanCell in C do
 4         initialize cellList with orphanCell
 5         nextCellWithEdges ← nil
 6         referenceCorner ← nil
 7         done ← false
 8         while ¬done do
 9             c ← last cell in cellList
10             cells, corner ← getCellsAtCorner(Q, c)
11             foreach cell in cells do
12                 nedges ← get edge count of cell in M
13                 if nedges > 0 then
14                     nextCellWithEdges ← cell
15                     referenceCorner ← corner
16                     done ← true
17                 else
18                     add cell to cellList
19                 end
20             end
21         end
22         foreach cell in cellList do
23             output(cell,nextCellWithEdges,
                 referenceCorner)
24             remove cell from C
25         end
26     end
27 end
```

Algorithm 2: getCellsAtCorner algorithm

```
   Input: a quadtree with cell envelopes Q and a cell c.
 1 function getCellsAtCorner(Q, c):
 2     region ← quadrant region of c in c.parent
 3     switch region do
 4         case 'SW' do
 5             corner ← left bottom corner of c.envelope
 6         case 'SE' do
 7             corner ← right bottom corner of c.envelope
 8         case 'NW' do
 9             corner ← left upper corner of c.envelope
10         case 'NE' do
11             corner ← right upper corner of c.envelope
12     end
13     cells ← cells which intersect corner in Q
14     cells ← cells − c
15     cells ← sort cells on basis of their depth
16     return (cells, corner)
17 end
```

Overlay optimizations

- Optimizing for faces overlapping many cells...
  - Naive approach sends all faces that overlap a cell to a master node (that will combine them).
  - We propose an intermediate reduce processing step.
    - The user provides a level in the quadtree structure and faces are evaluated at those intermediate reducers.
  - We also consider another approach that re-partitions such faces using their labels as the key.
    - It avoids the reduce phase but implies an additional shuffle.
    - However, as we show in the experiments this overhead is minimal.

Overlay optimizations

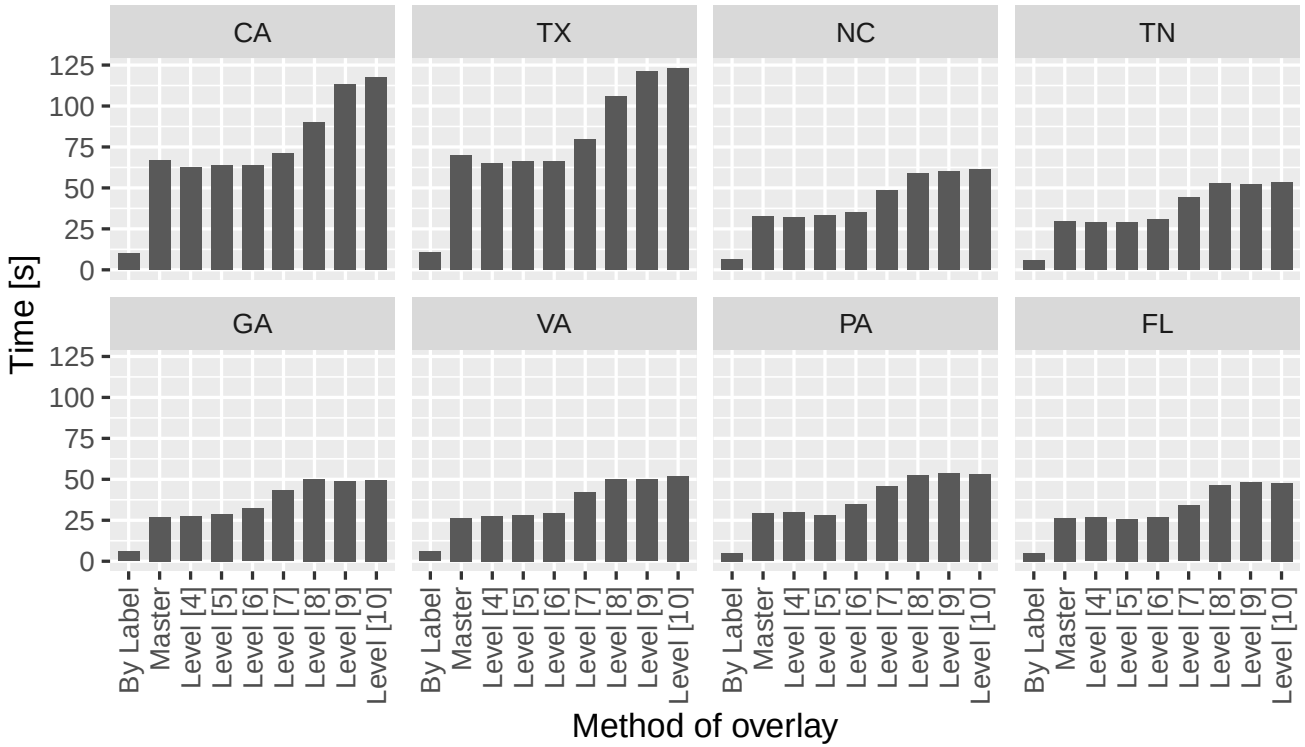
- Optimizing for unbalanced layers...
  - Finding intersections is the most critical part of the overlay computation.
  - However, in many cases one of the layers has much more half-edges than the other.
  - Sweep-line algorithms to detect intersections run over all the edges.
  - Instead we scan the larger dataset only for the x-intervals where there are half-edges from the smaller dataset.

- Datasets.

Dataset	Layer	Number of polygons	Number of edges
MainUS	Polygons for 2000	64983	35417146
	Polygons for 2010	72521	36764043
GADM	Polygons for Level 2	116995	32789444
	Polygons for Level 3	117891	37690256
CCT	Polygons for 2000	7028	2711639
	Polygons for 2010	8047	2917450

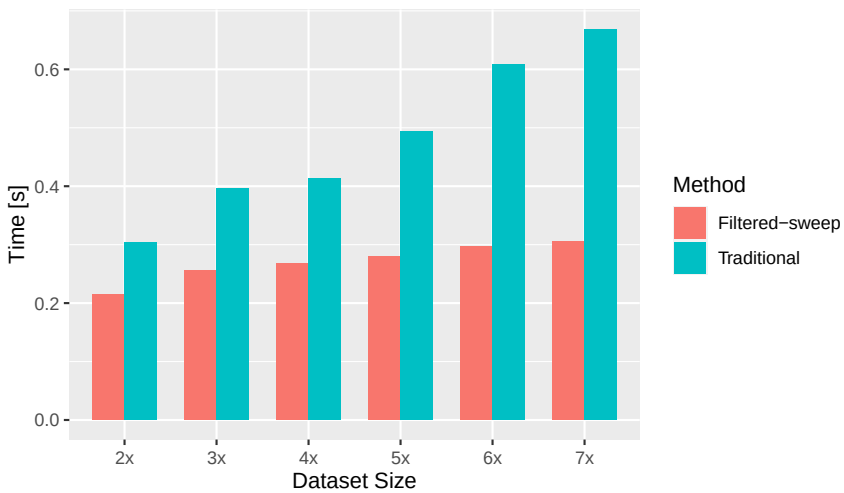
Experimental evaluation

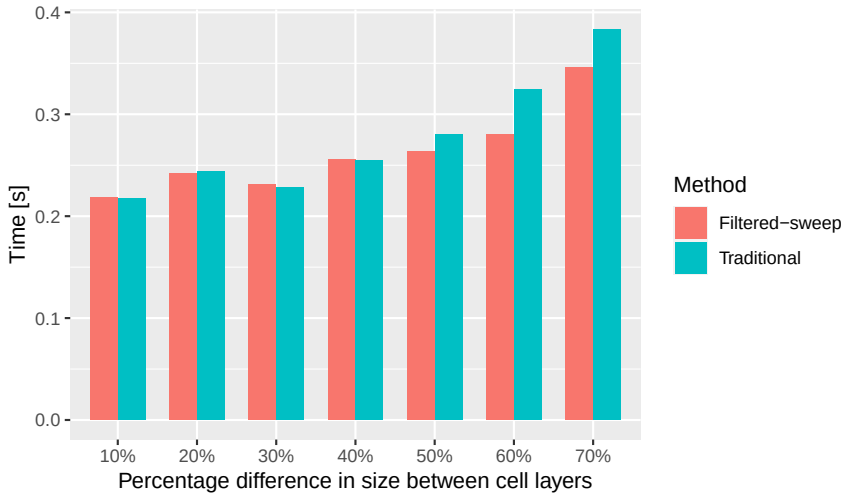
- Evaluation of the overlapping faces optimization.



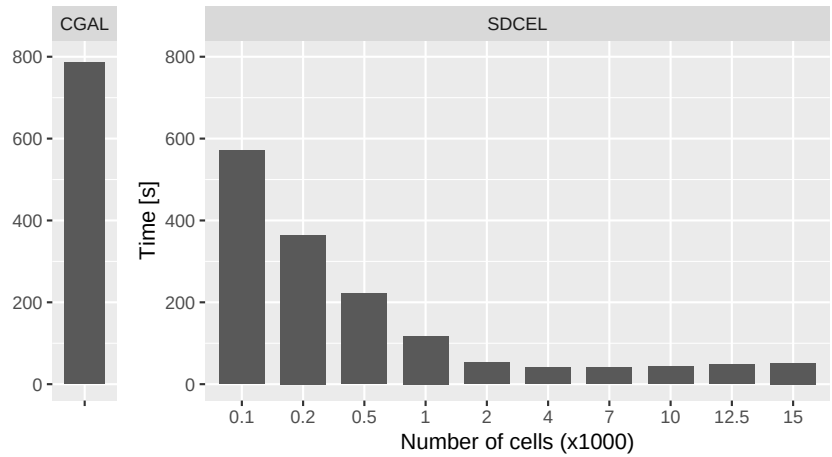
Experimental evaluation

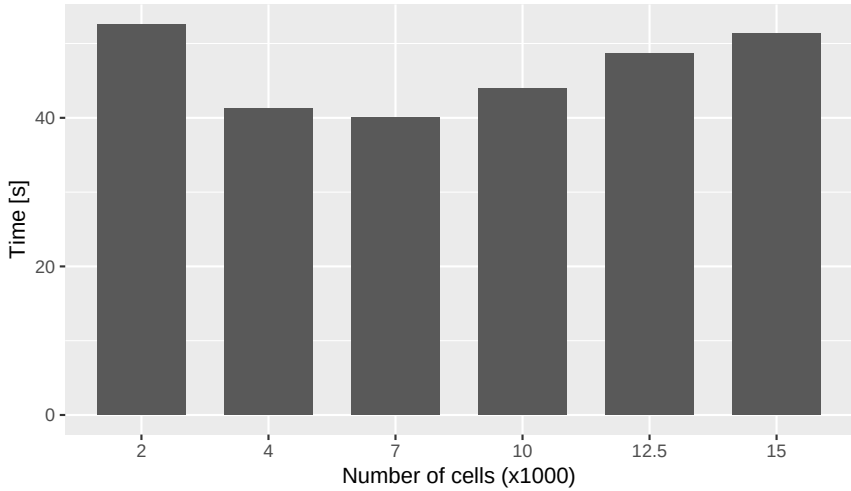
- Evaluation of the unbalanced layers optimization.



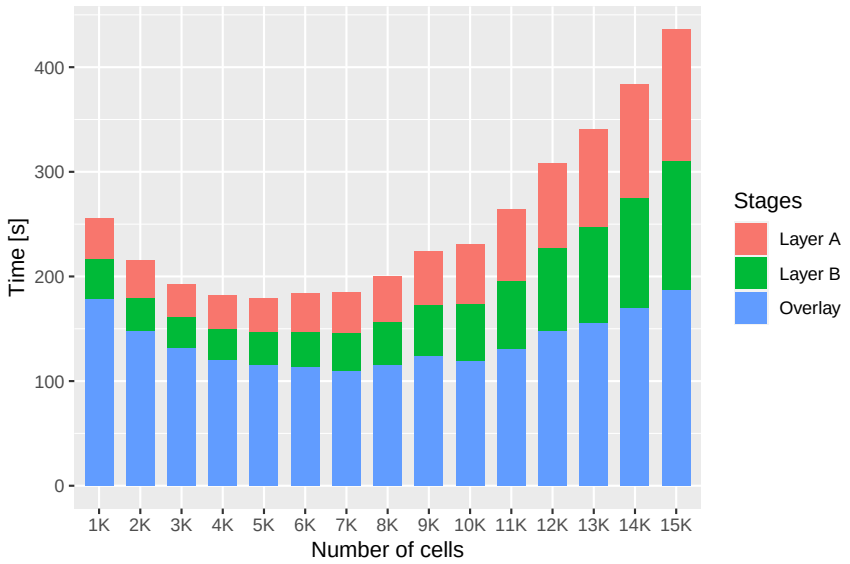


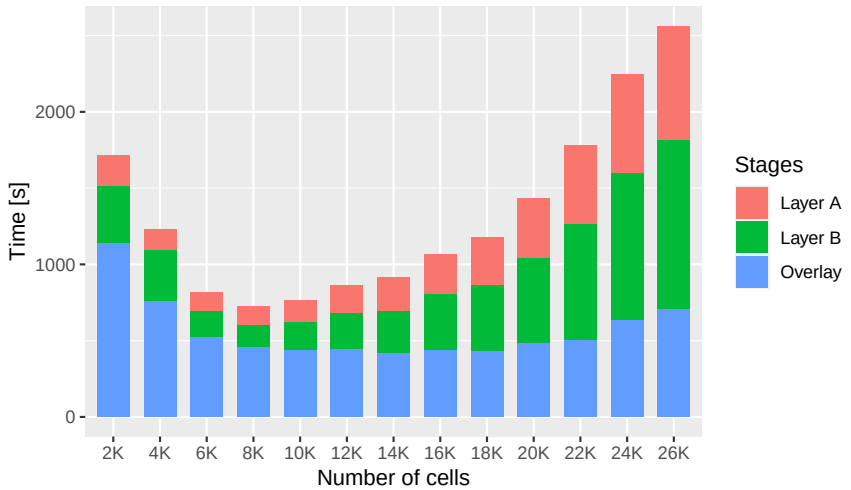
- Performance varying number of partition cells (CCT dataset).



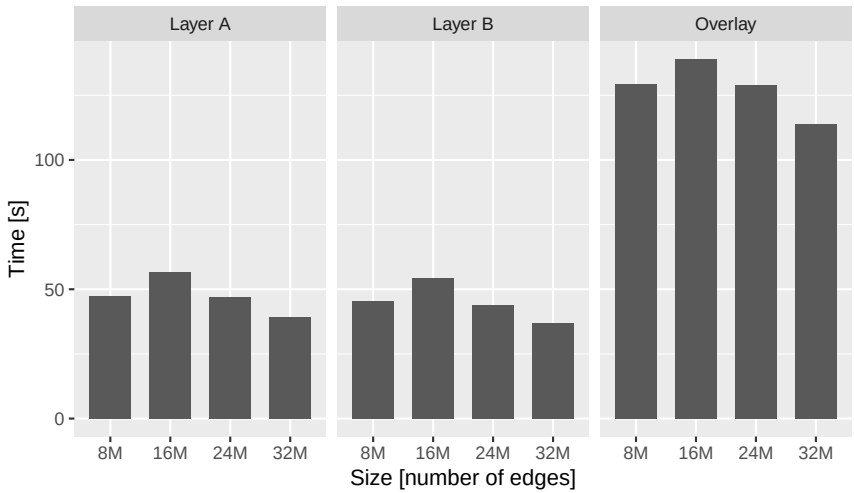


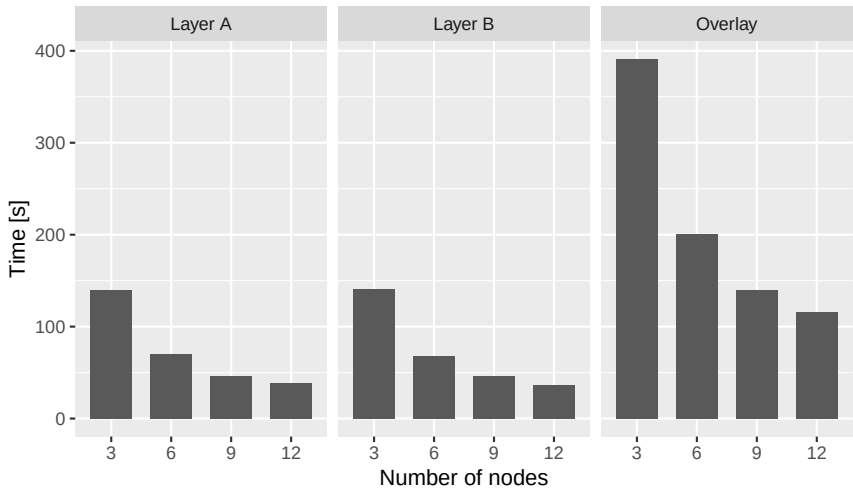
- Performance with MainUS and GADM datasets.





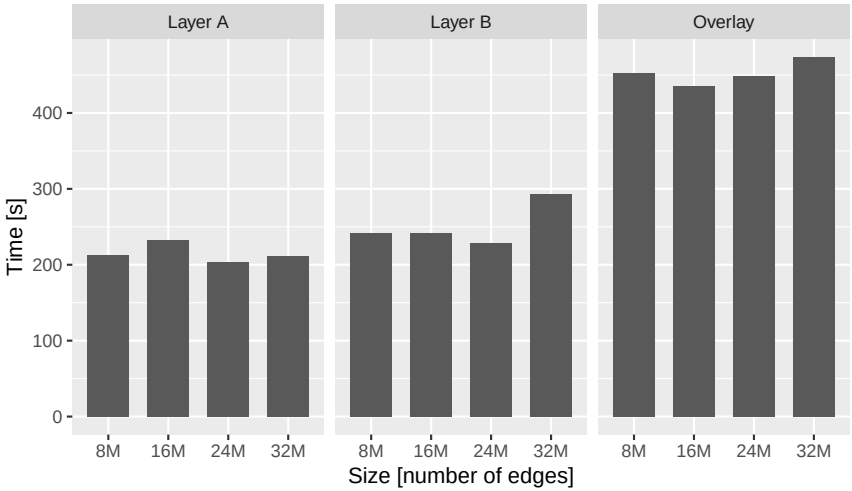
- MainUS scale-up and speed-up.

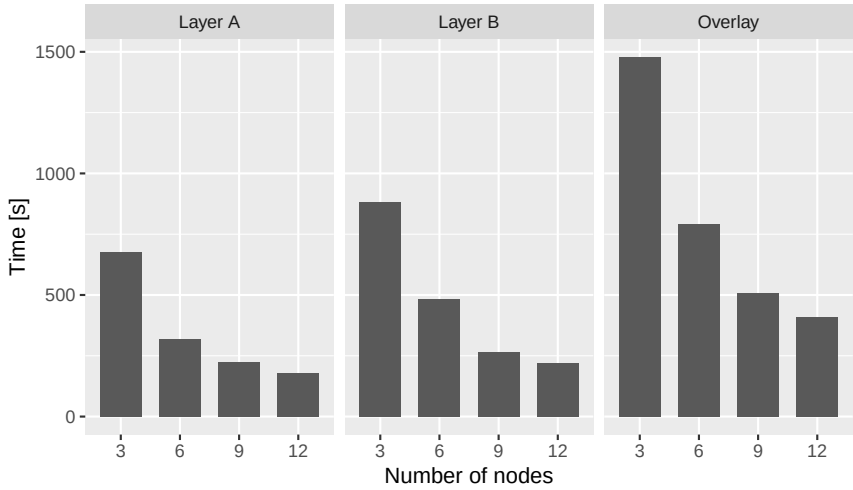




Experimental evaluation

- GADM scale-up and speed-up.





Conclusions

- We introduced SDCEL, a scalable approach to compute the overlay operation among two layers that represent polygons from a planar subdivision of a surface.
- We use a partition strategy which guarantees that each partition (cell) has the data needed to work independently.
- We also proposed several optimizations to improve performance.
- Our experiments using real datasets show very good performance; we are able to compute overlays over very large layers (each with >35M edges) in few minutes.

Outline

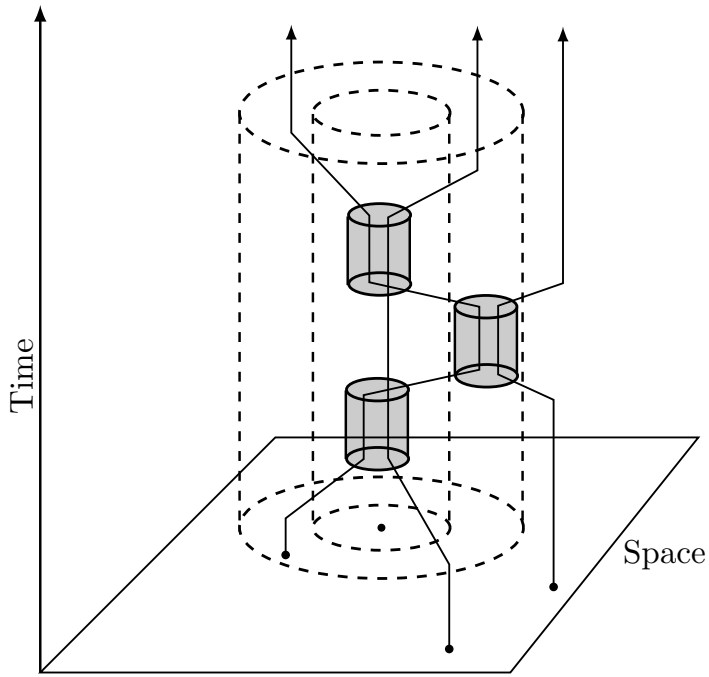
- Scalable overlay operations over DCEL polygons layers
- **An Extension On Scalable DCEL Overlay Operations**
- Scalable Processing of Moving Flock Patterns

Outline

- Scalable overlay operations over DCEL polygons layers
- An Extension On Scalable DCEL Overlay Operations
- **Scalable Processing of Moving Flock Patterns**

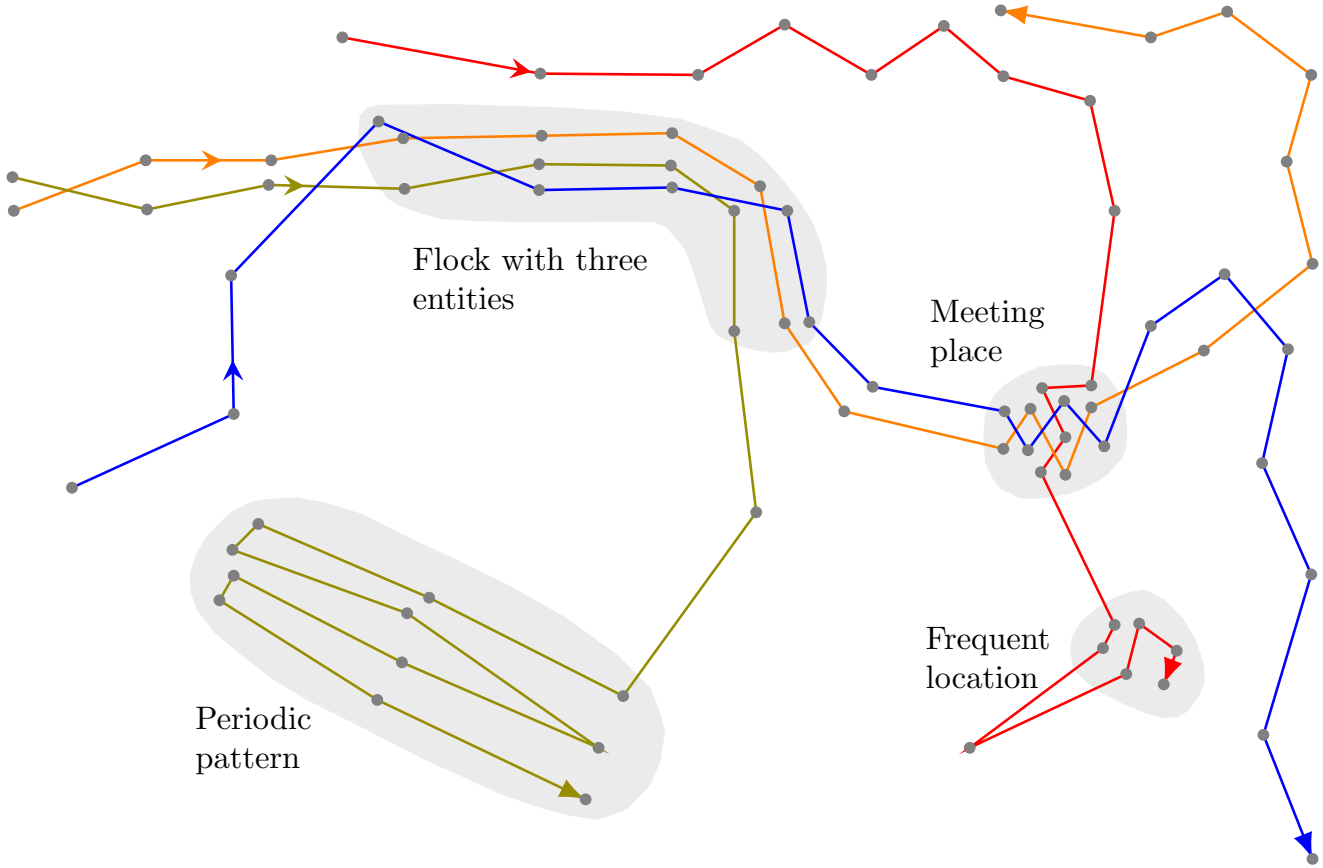
Large trajectory databases

- A spatial trajectory is a trace in time generated by a moving entity in a geographical space.
- i.e. $p_1 \rightarrow p_2 \rightarrow \cdots \rightarrow p_n$
- A trajectory is stored as a time-ordered sequence of points, $p_i = (x, y, t)$ (spatial coordinate + time stamp).



(Shoval, 2017)

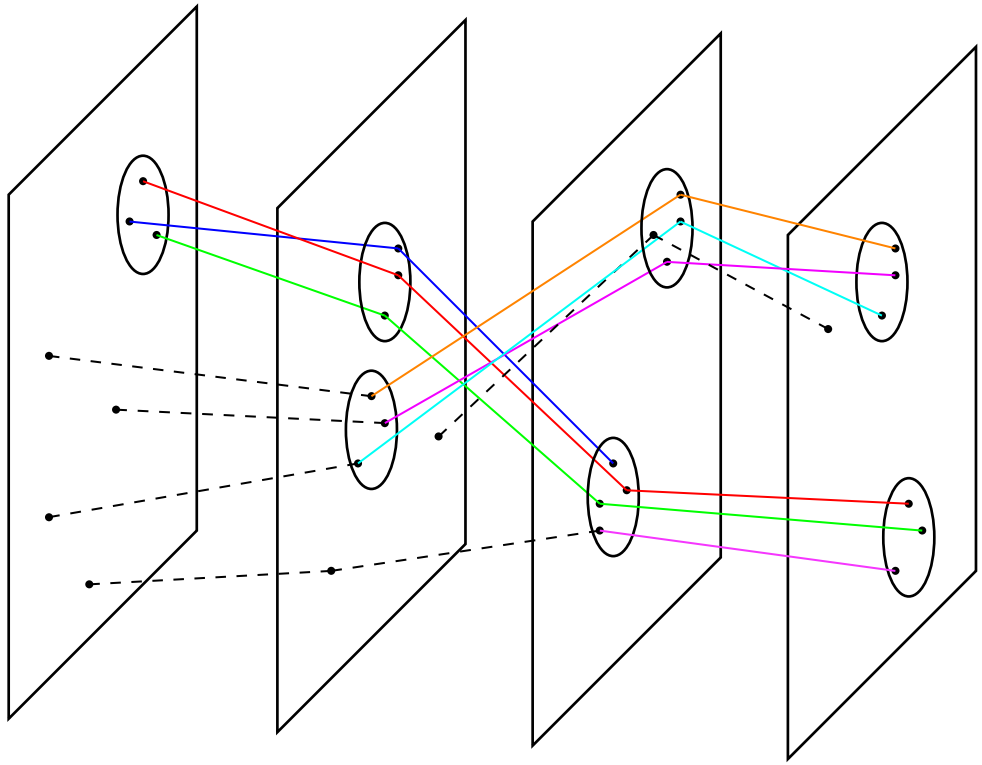
Movement patterns



(Gudmundsson, et al. 2008)

- i.e. convoys, moving clusters, swarms, gatherings, **flocks**, ...

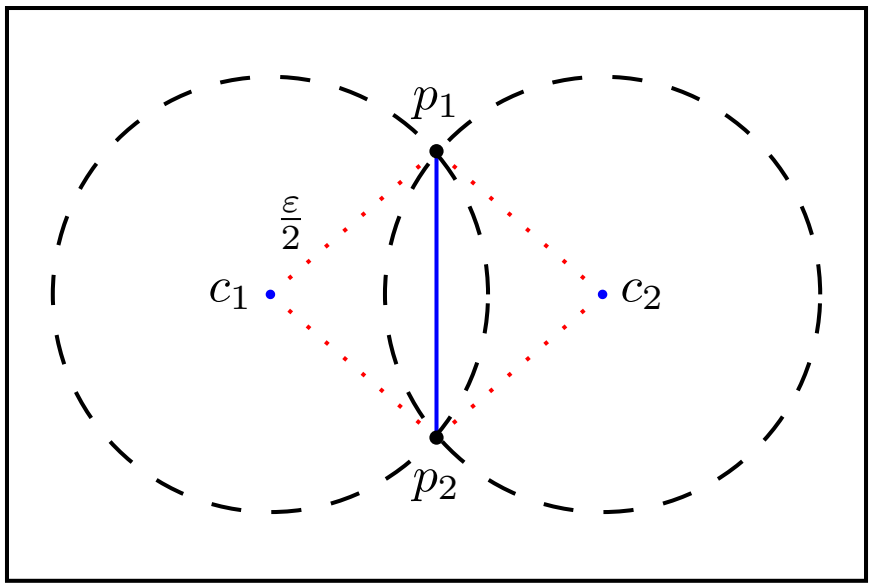
Flocks



- ε: Diameter of the circle which contains all the objects.
- μ: Minimum number of objects.
- δ: Minimum time interval the objects travel 'together'.

Basic Flock Evaluation algorithm

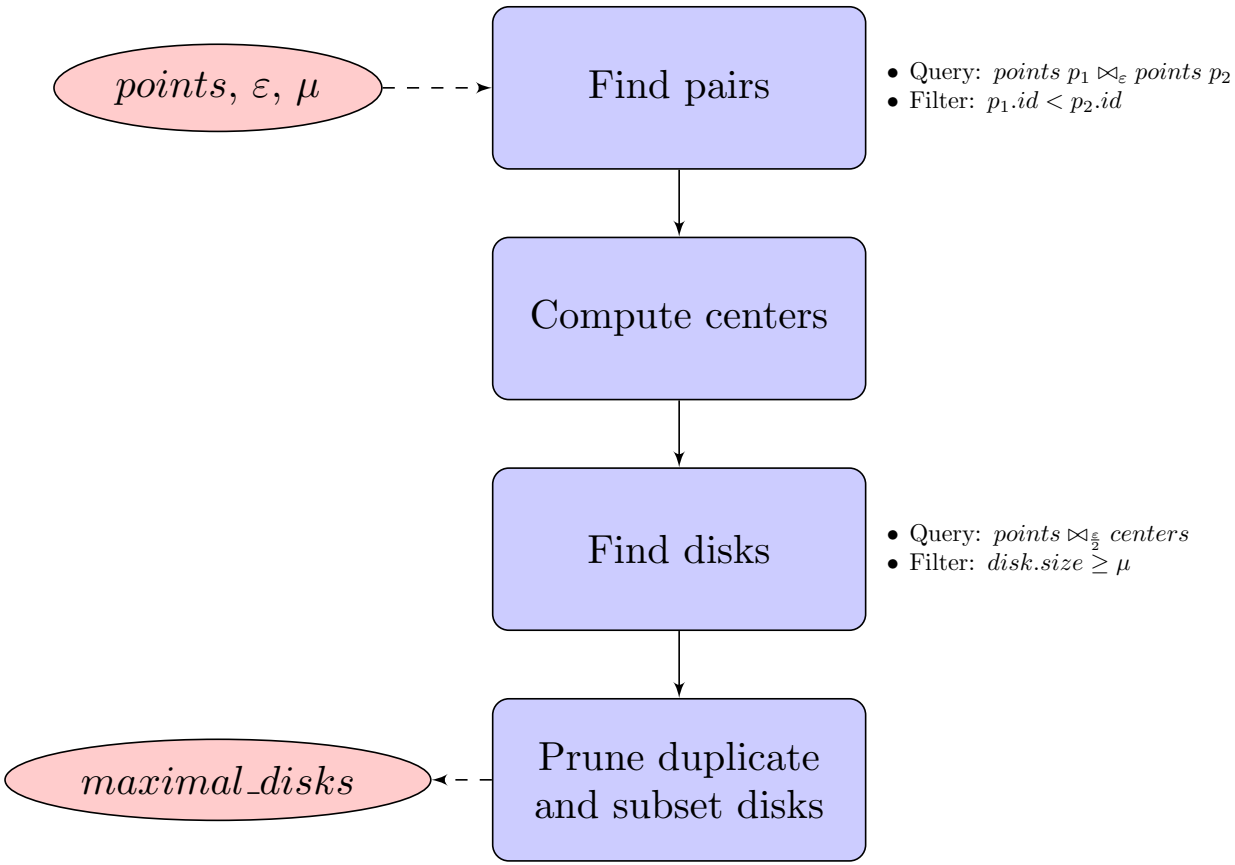
- Vieira, et al. 2009.
- The first polynomial-time solution for determining disk locations.
- Under fixed time duration it has polynomial time complexity $O(\delta|\tau|^{(2\delta)+1})$



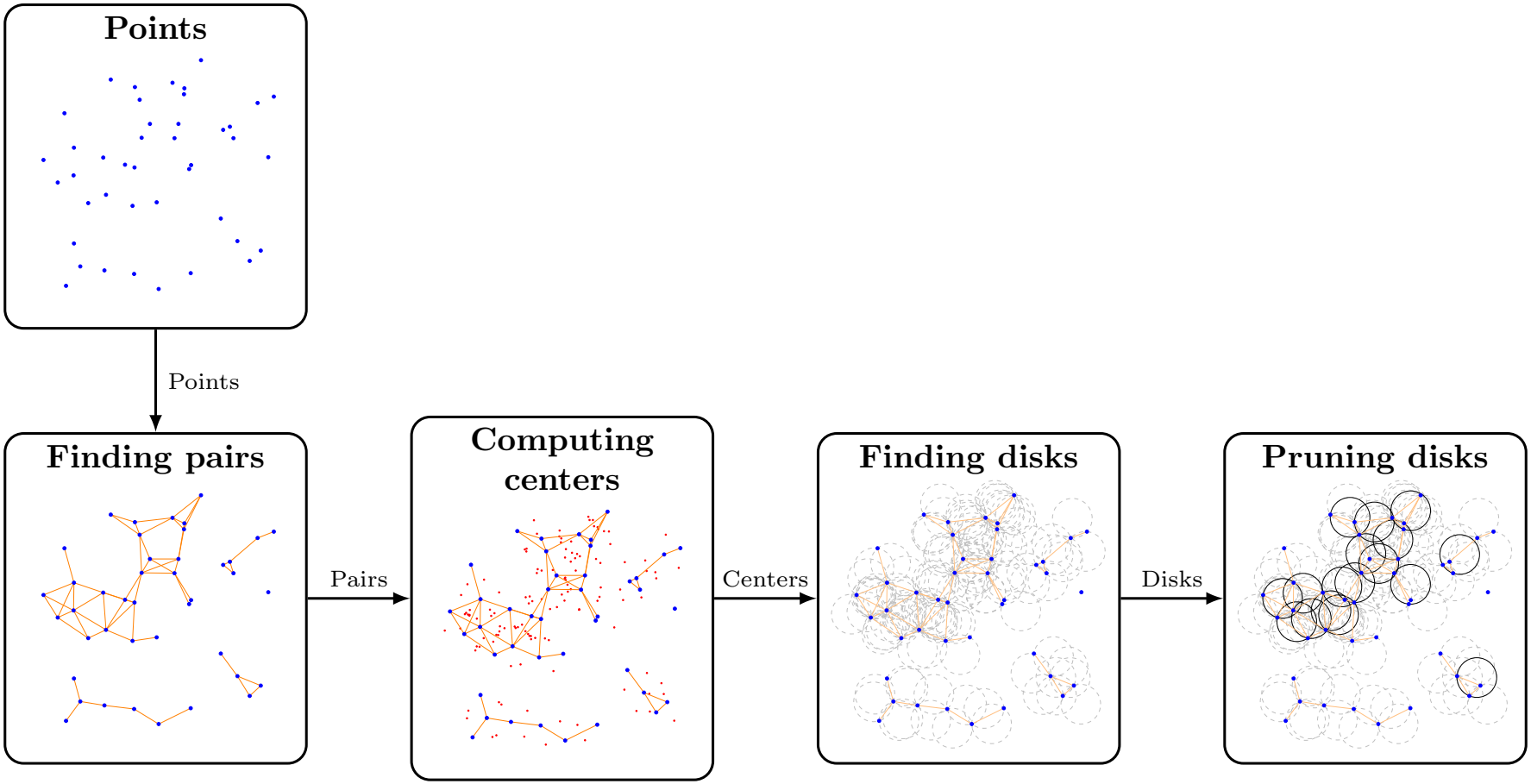
Basic Flock Evaluation algorithm

- Two main parts:
  - In the spatial domain it finds maximal disks at each time instant.
  - In the temporal domain it joins consecutive times to match set of maximal disks.

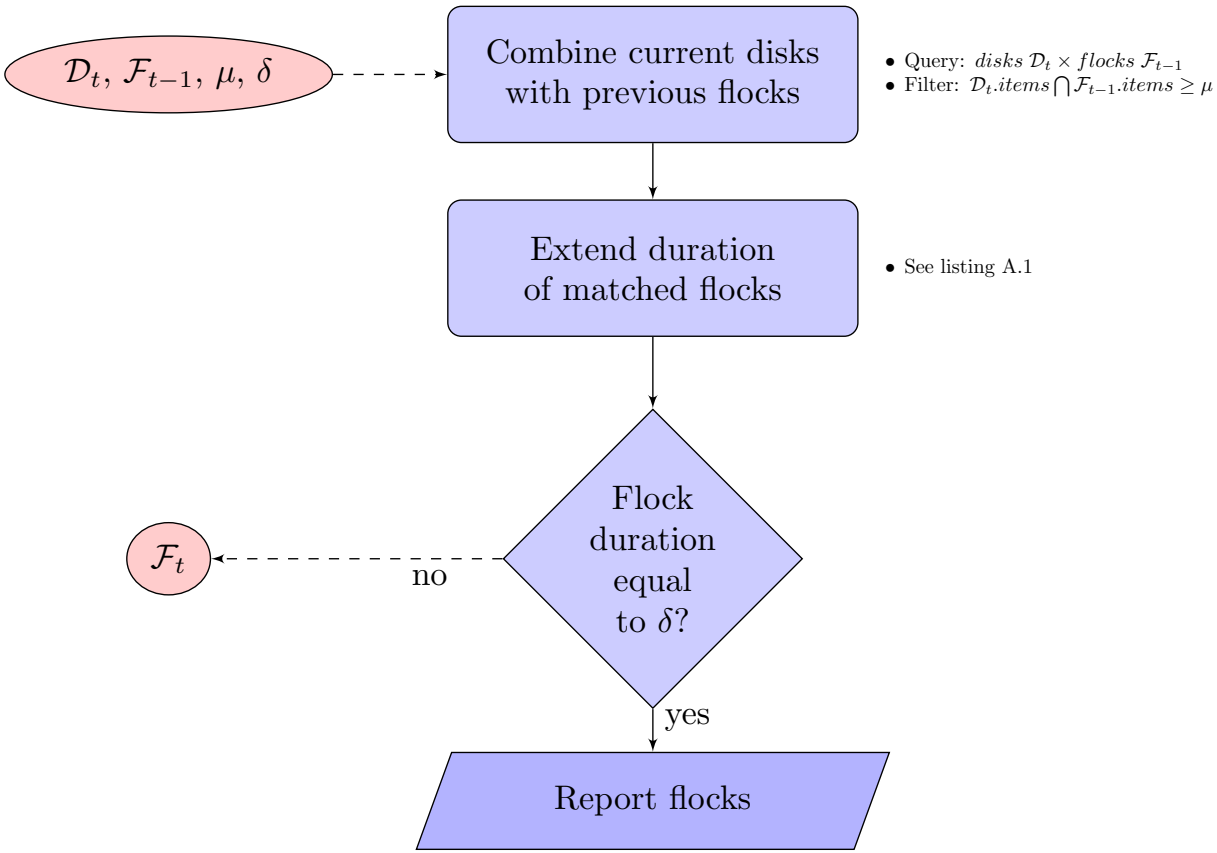
- BFE overview...



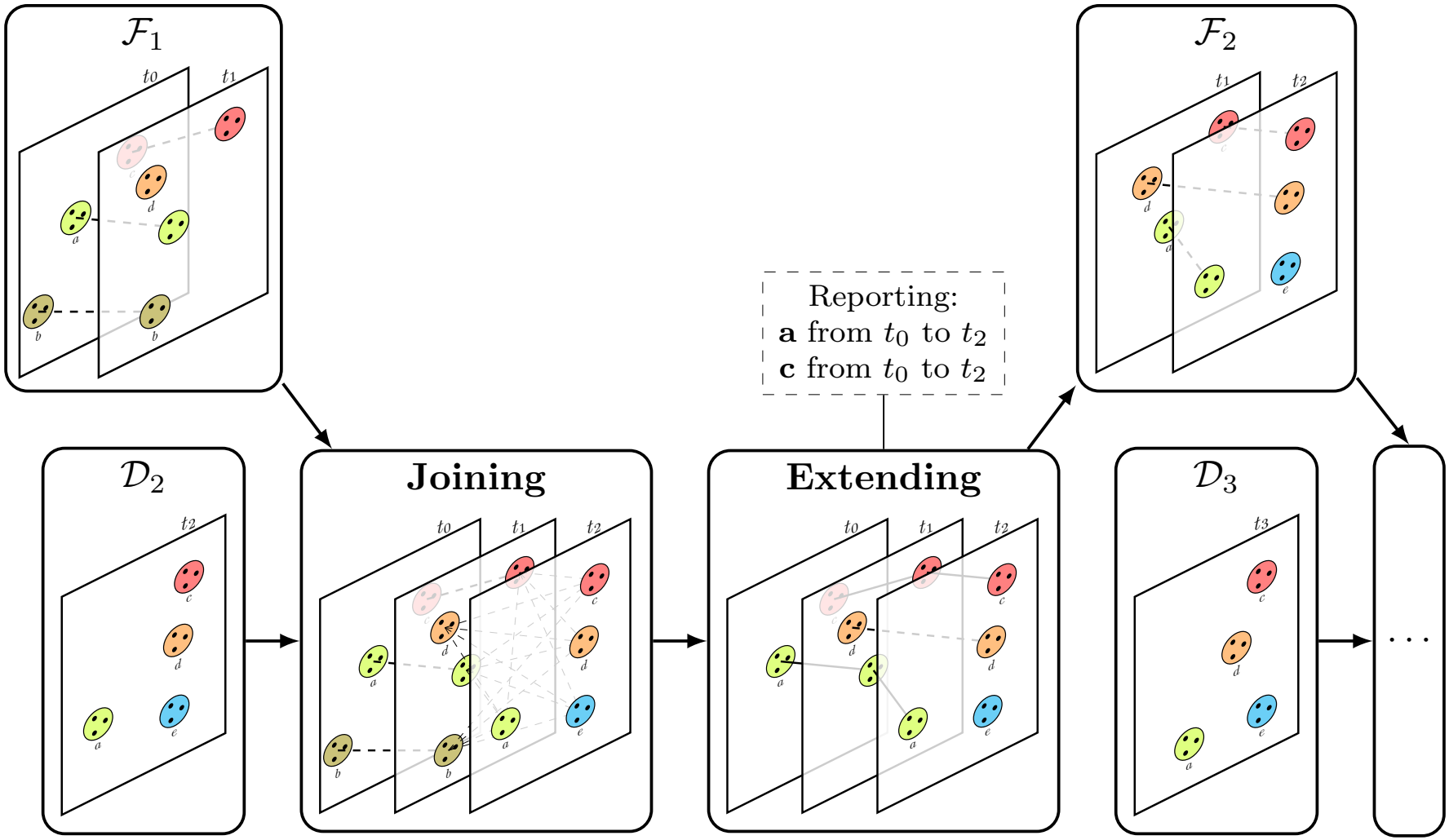
- BFE overview...



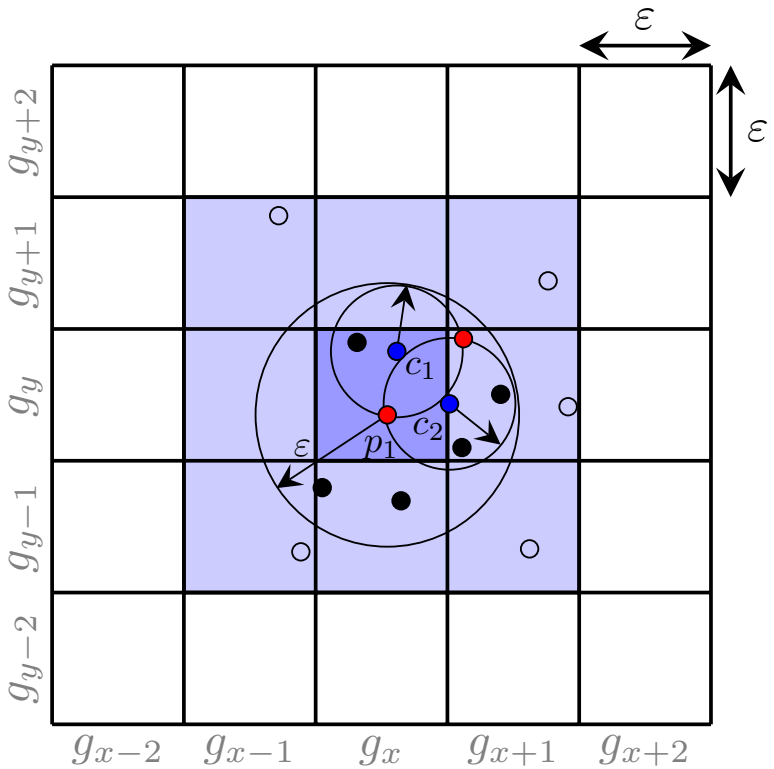
- BFE overview...



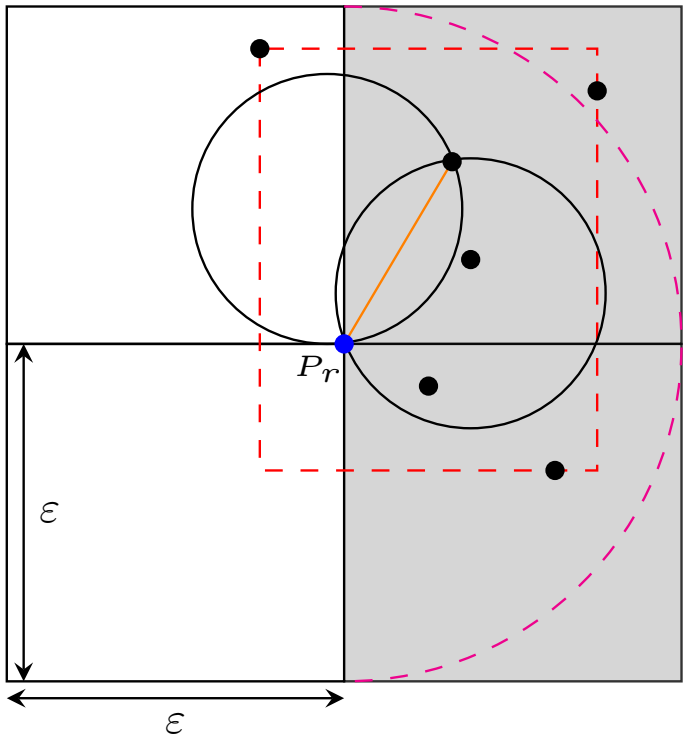
- BFE overview...



PSI ALGORITHM



(Vieira, et al. 2009)

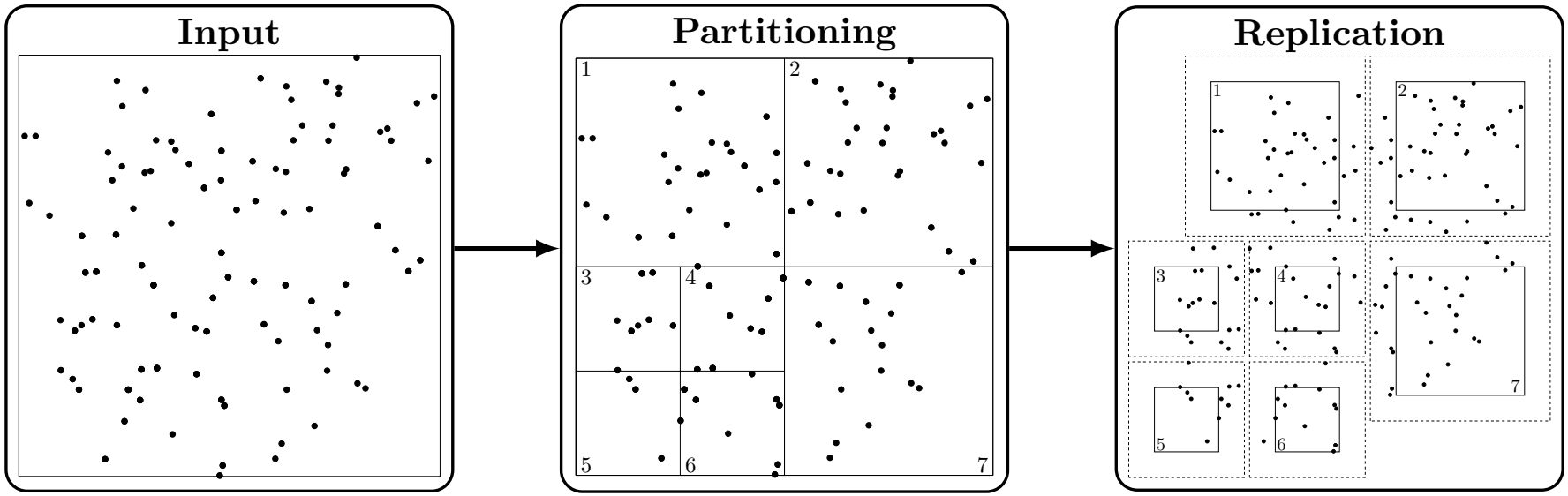


(Tanaka, et al. 2016)

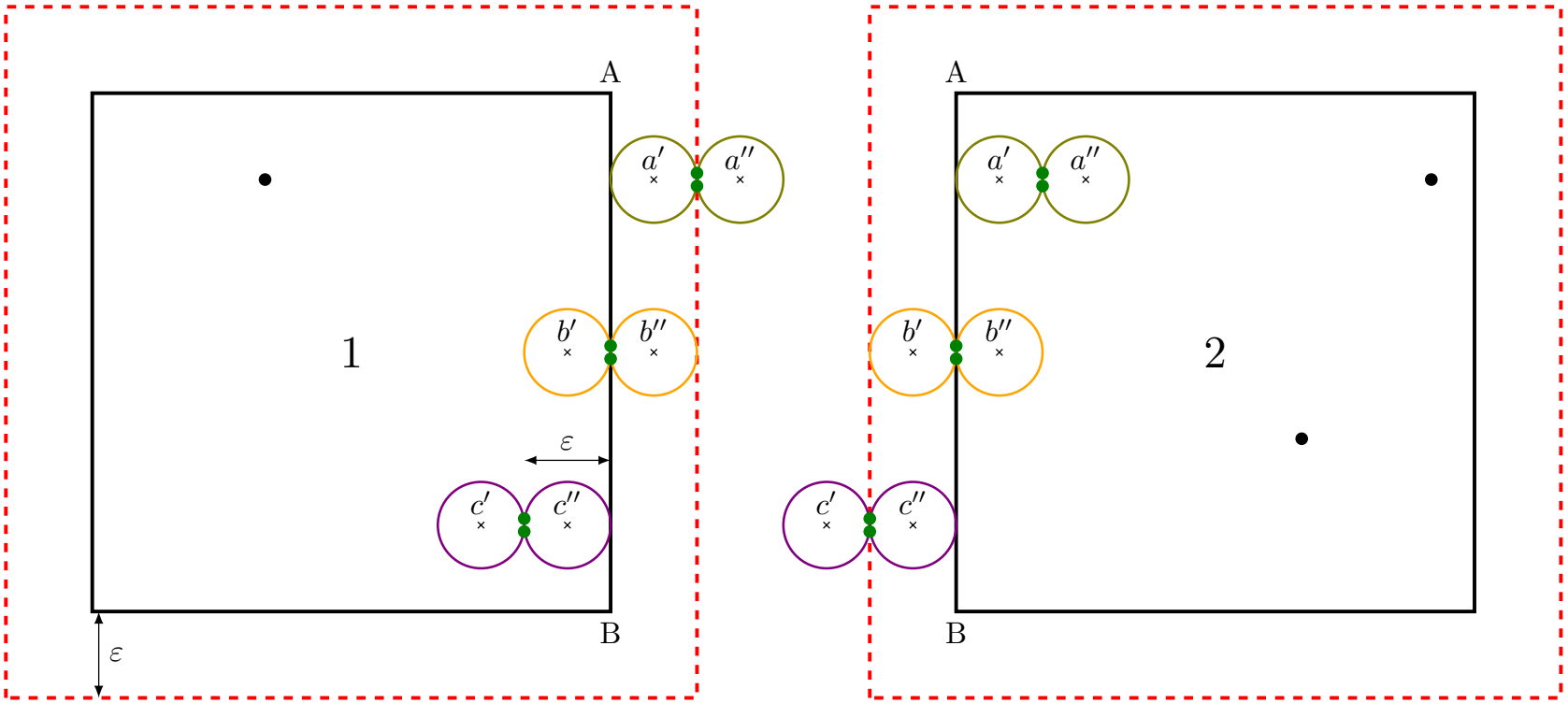
Challenges and contributions

- High complexity limits scalability.
- Large datasets with dense clusters of moving entities per time instant significantly impact performance.
- Specifically,
  - identifying maximal disks is hindered by the extensive number of candidates requiring pruning.
  - when parallelizing, we must address moving flocks that traverse contiguous partitions.
- We propose a parallel and scalable solution for both spatial and temporal domains.

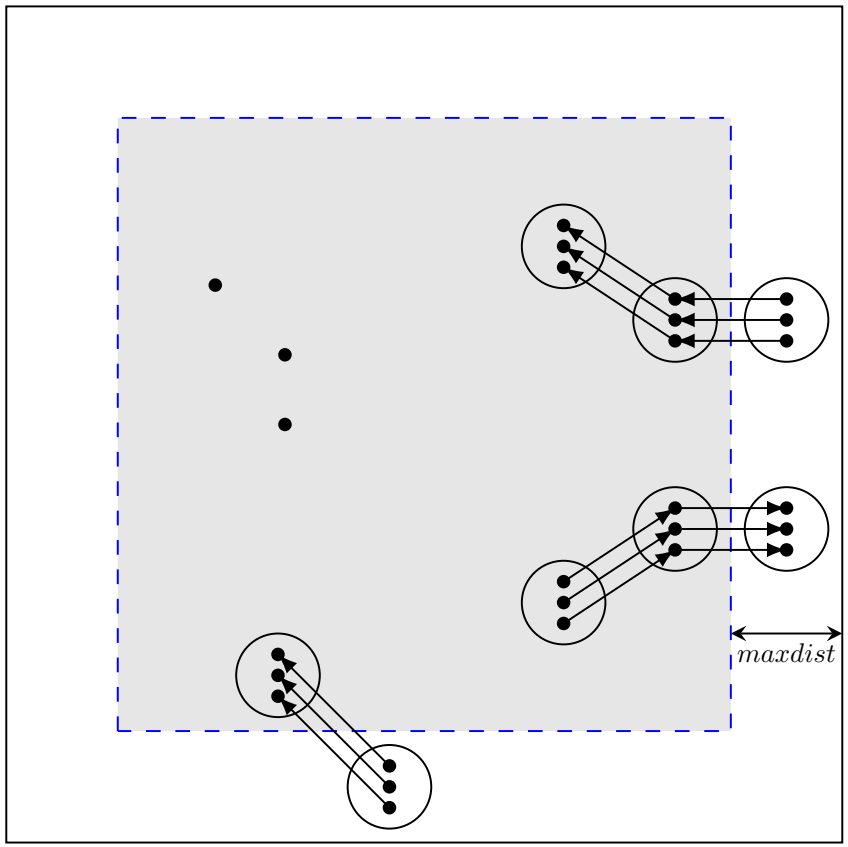
- Partitioning strategy...



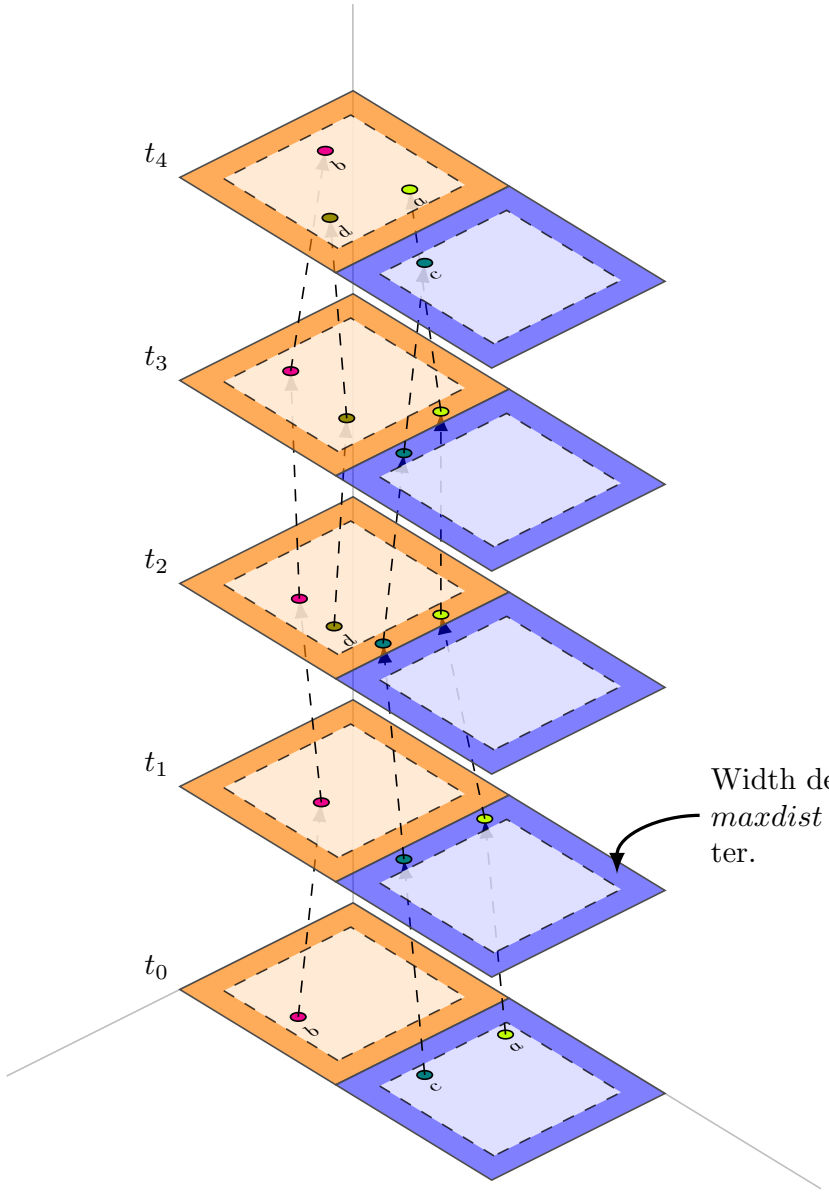
- Handling duplication...



- We introduce the *maxdist* parameter to define an area were we have to track **crossing partial flocks** (CPFs)...



On the temporal domain

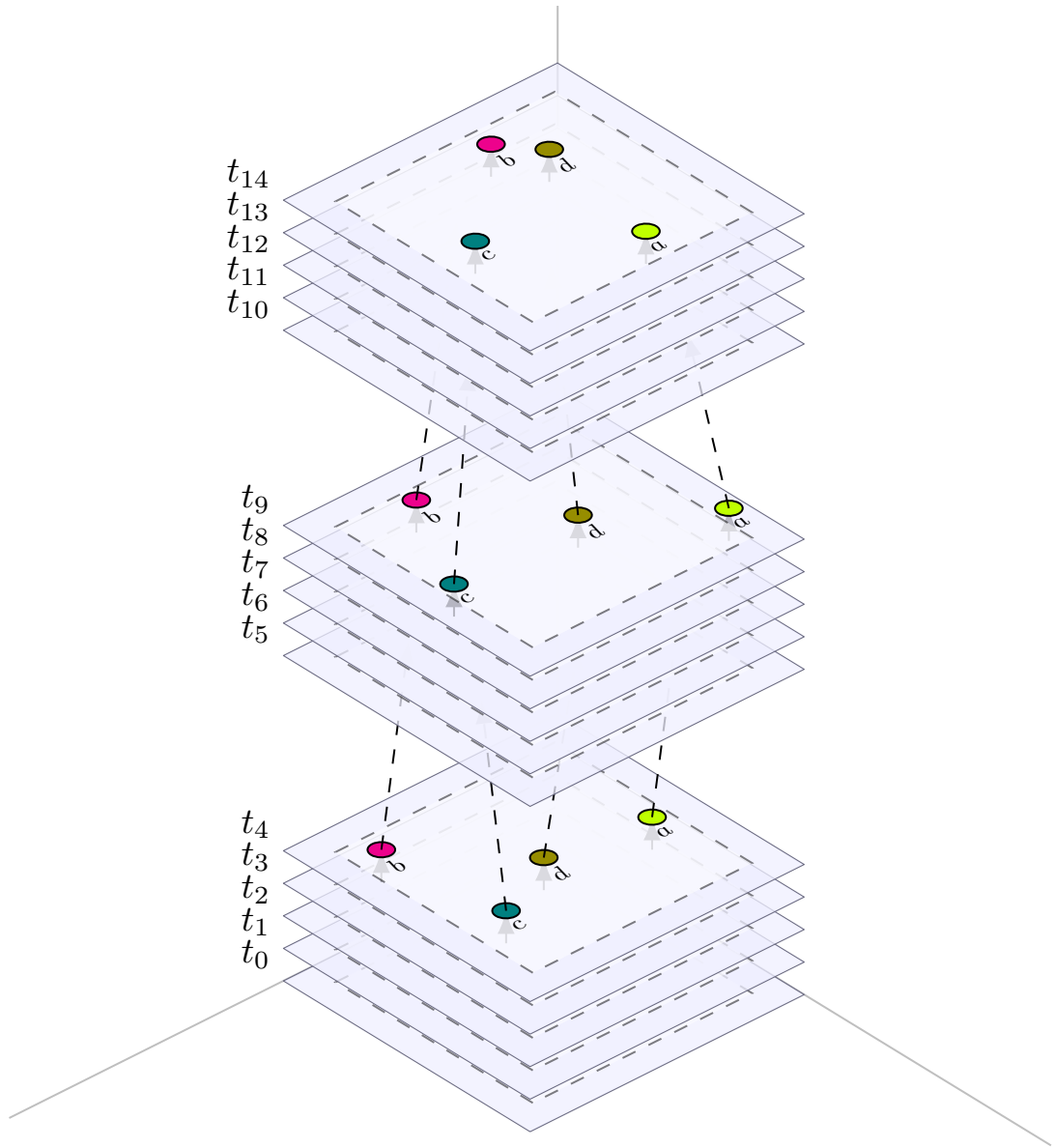


*a,b,c and d are flocks moving along time.

On the tempora domain

- Discovered flocks inside the safe area are ready to be reported.
- CPFs require post-processing. We propose four alternative:
  - Master
  - By-Level
  - Least Common Ancestor (LCA)
  - Cube-based

On the temporal domain

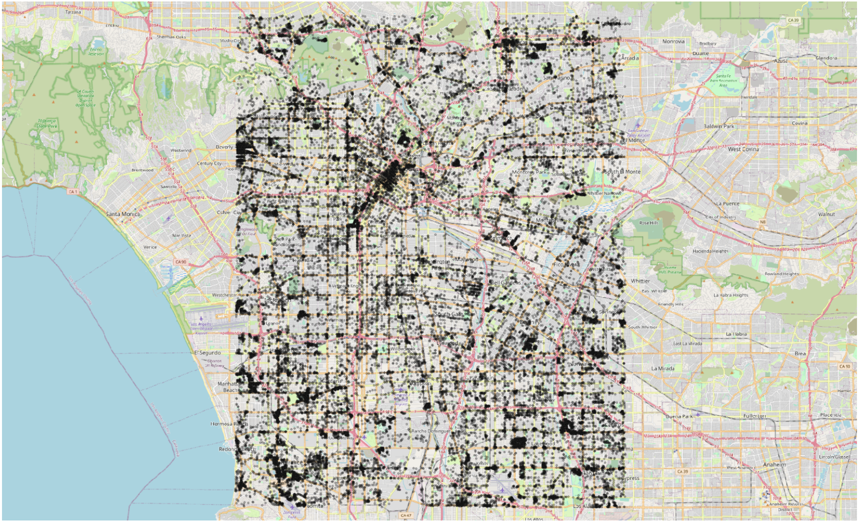


*a,b,c and d are flocks moving along time.

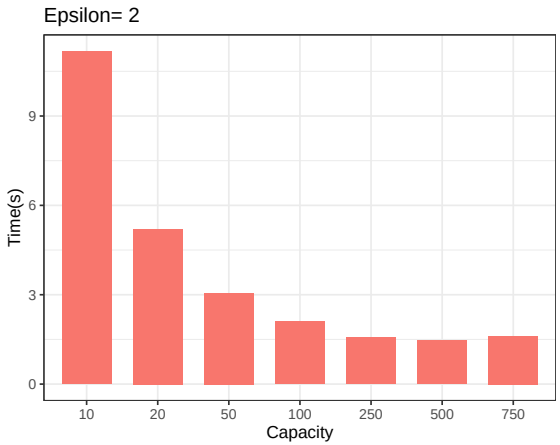
Datasets

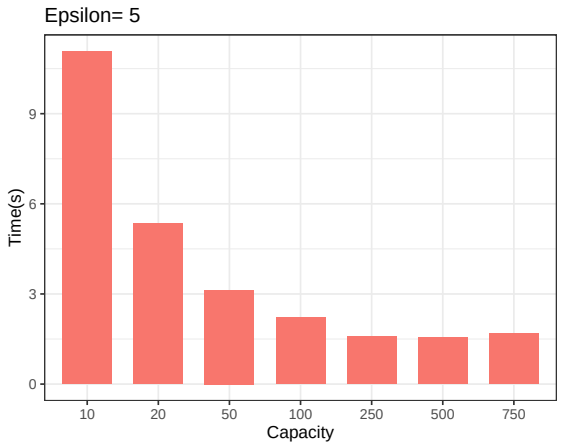
Dataset	Number of Trajectories	Total number of points	Maximum Duration (min)
Berlin10K	10000	97526	10
LA25K	25000	1495637	30
LA50K	50000	2993517	60

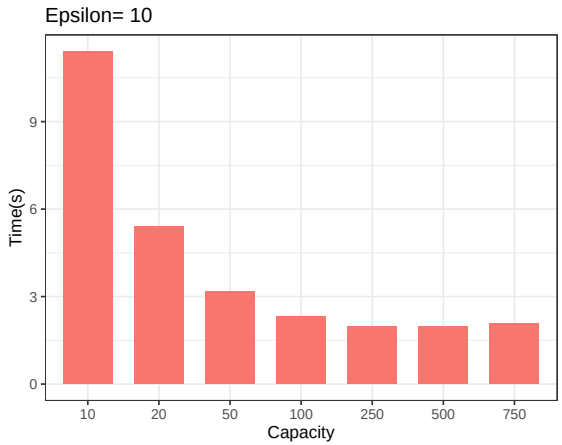
- Synthetic datasets [LA50K]

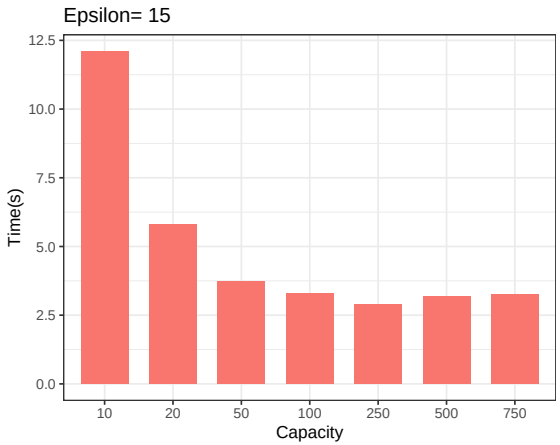


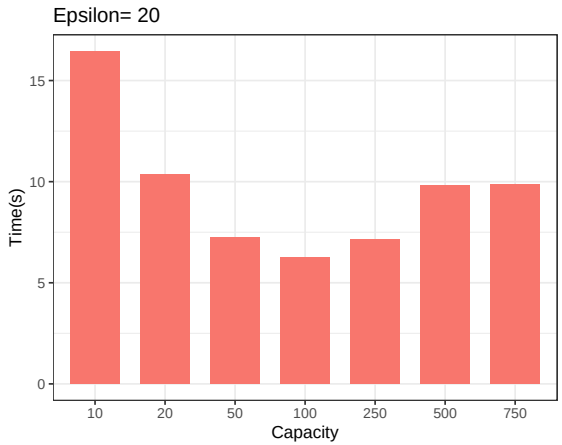
- Optimizing the number of partitions for Phase 1.





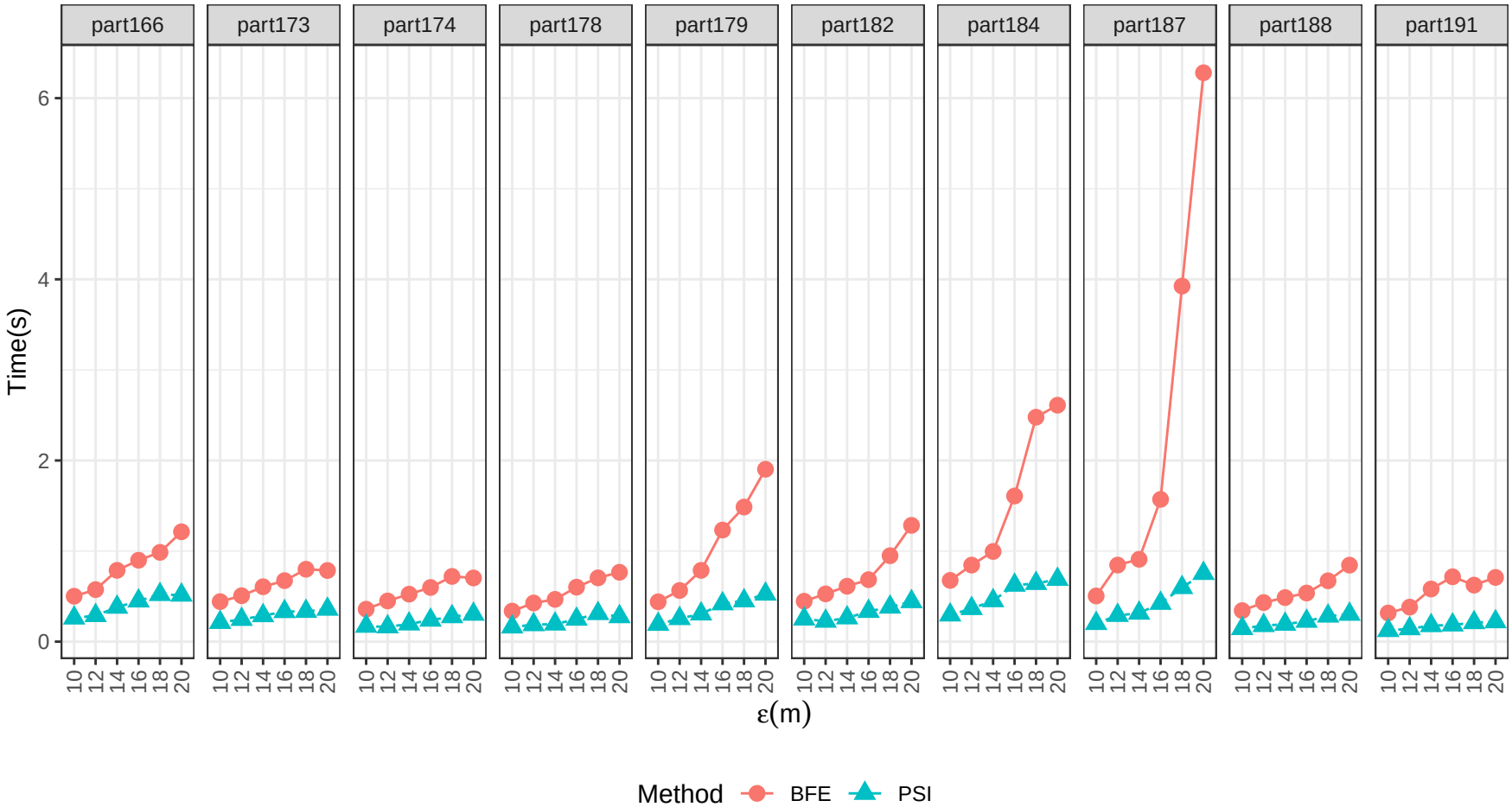




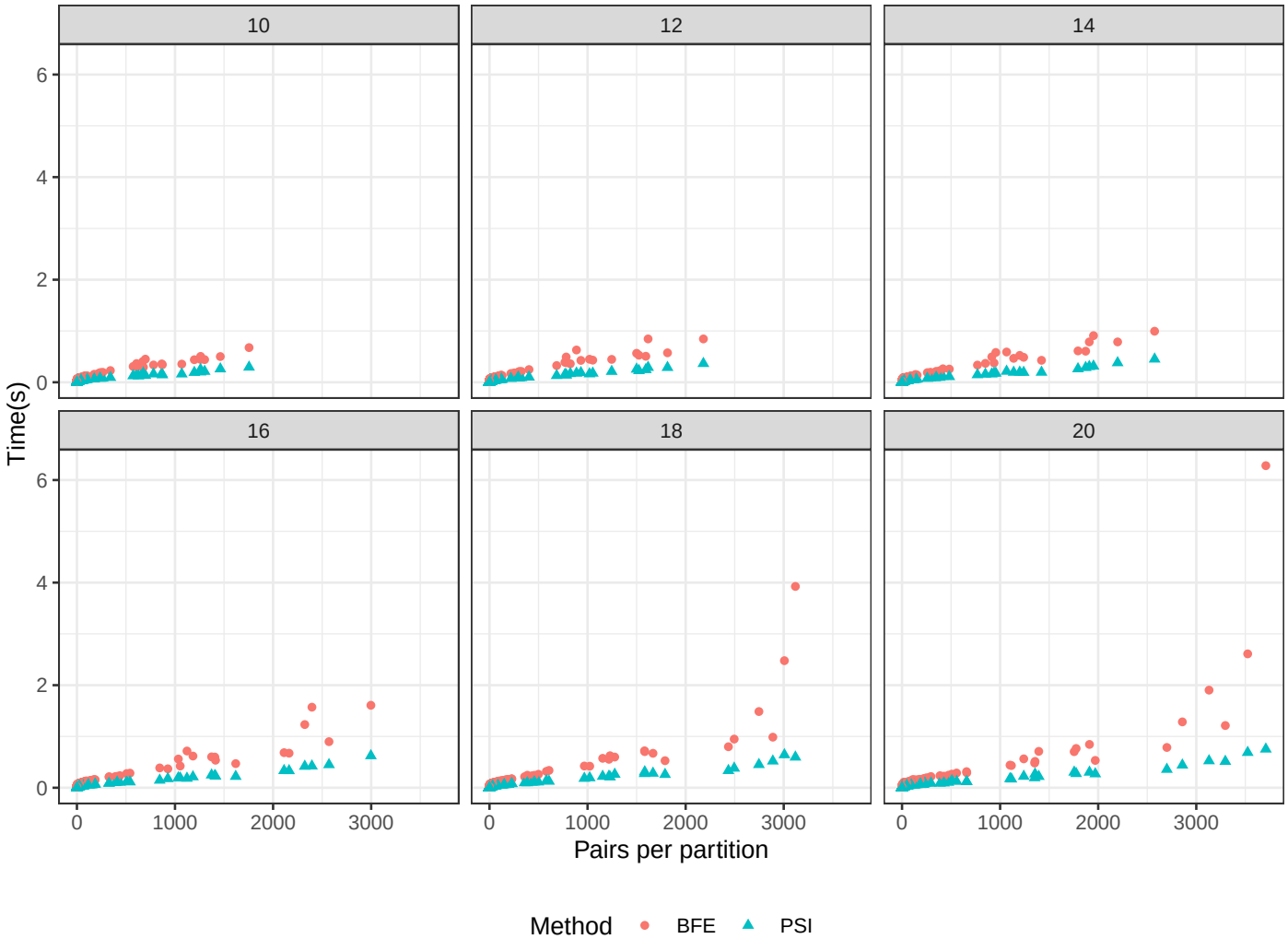


Experiments

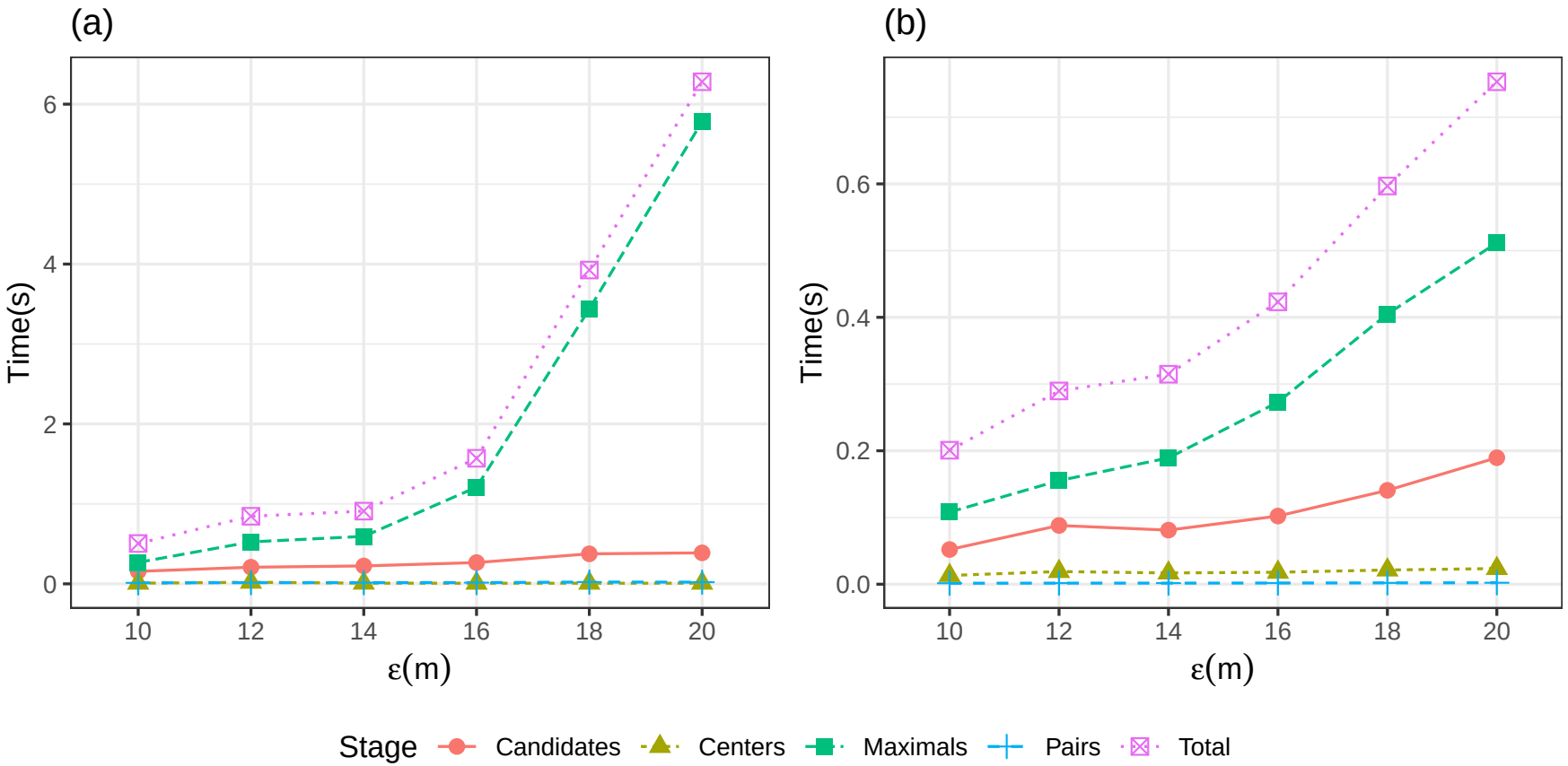
- Analyzing most costly partitions.
  - Top 10 most costly partitions.



- Analyzing most costly partitions.
  - By Pairs density..



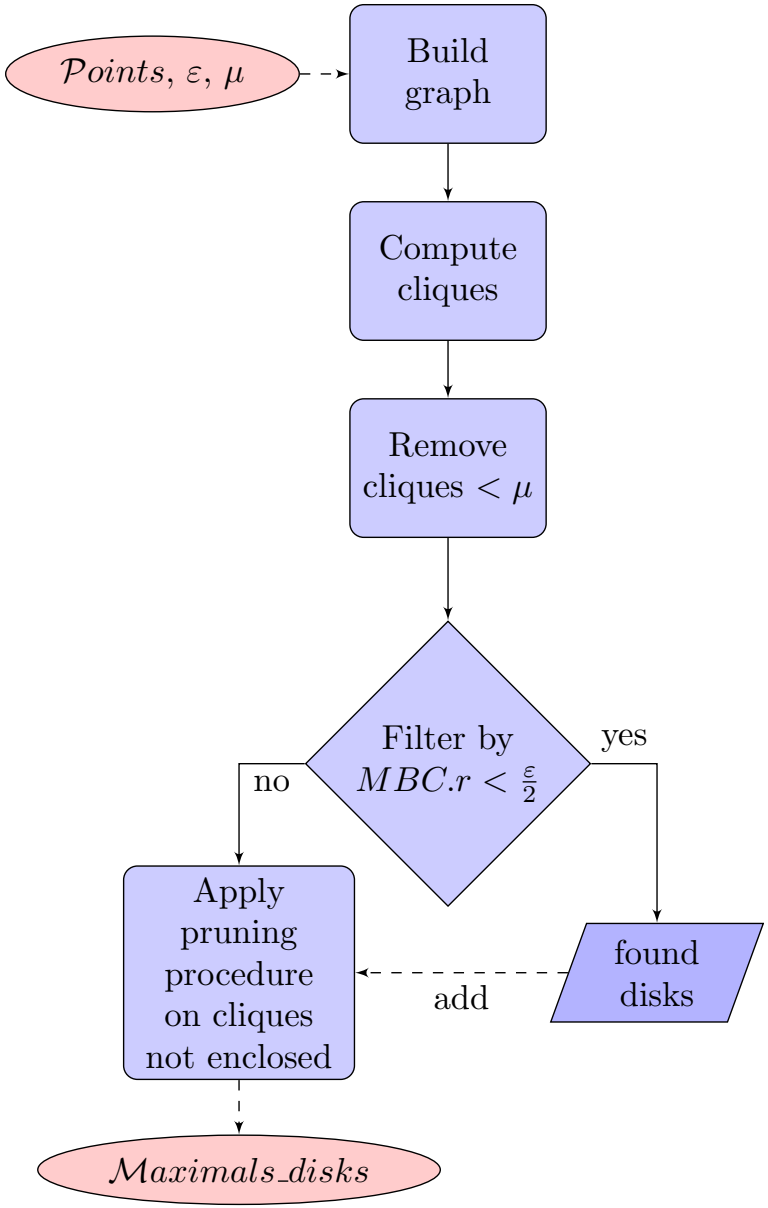
- Analyzing most costly partitions.
  - By Stages in the most costly partition [(a) BFE (b) PSI].



Can we reduce pruning time?

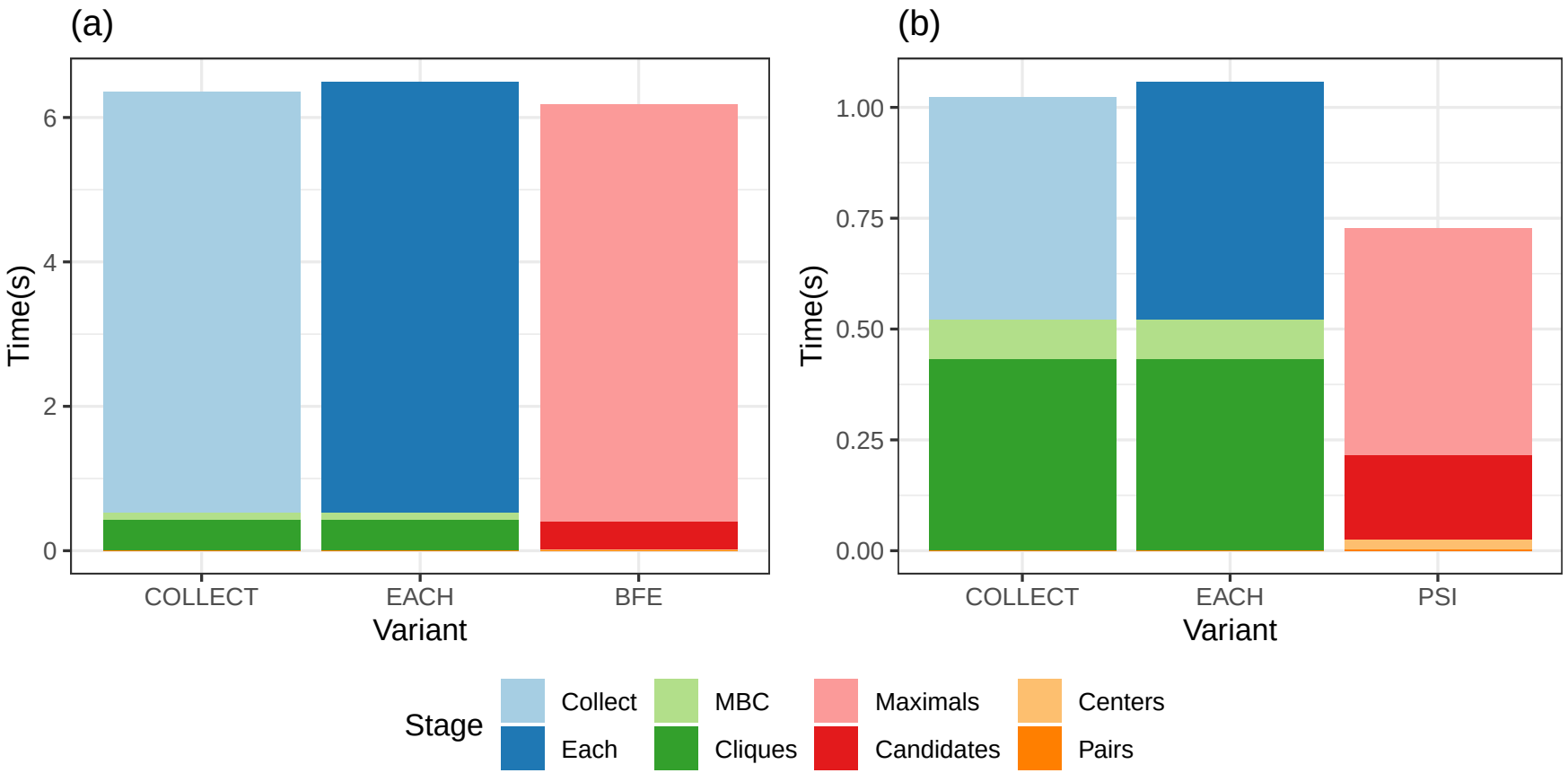
- **Maximal clique (MC)**: Given an undirected graph, a MC is a subset of vertices, each directly connected to every other in the subset, that cannot be expanded by adding additional vertices.
- **Minimum Bounding Circle (MBC)**: Given a set of points in Euclidean space, the MBC is the smallest circle that can enclose all the points.

Can we reduce pruning time?

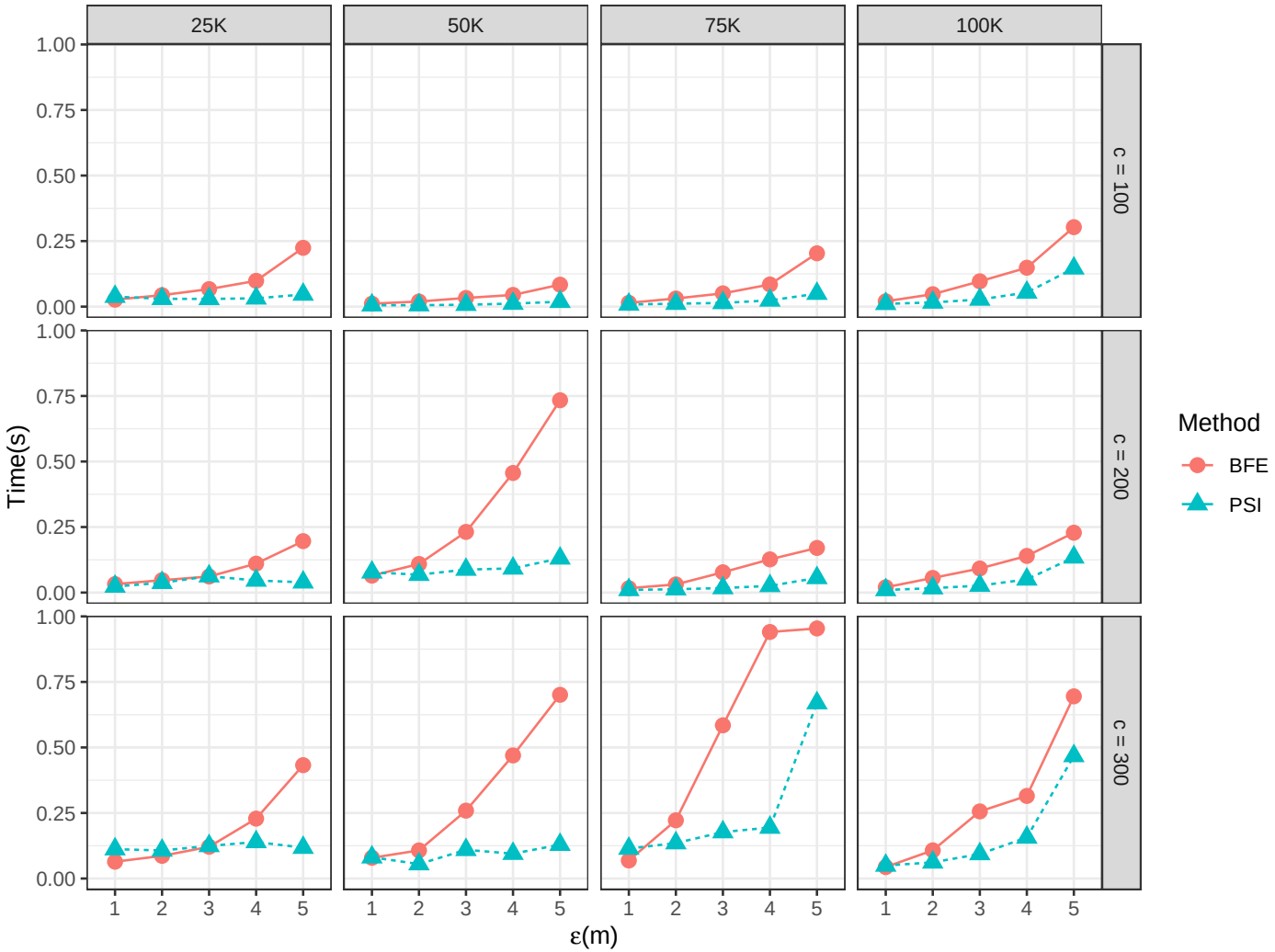


Can we reduce pruning time?

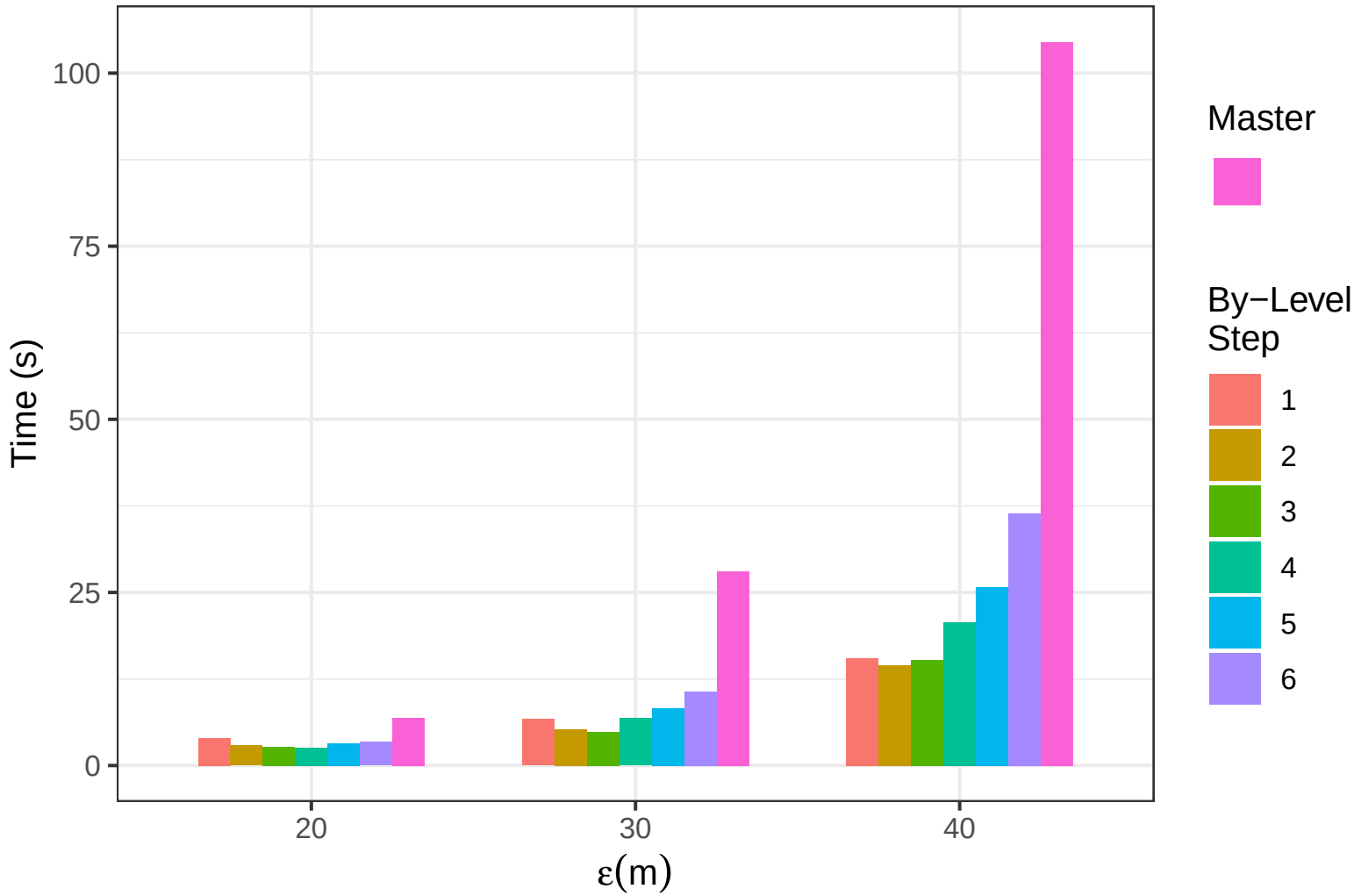
- Phase 1 variants performance [(a) vs BFE (b) vs PSI].



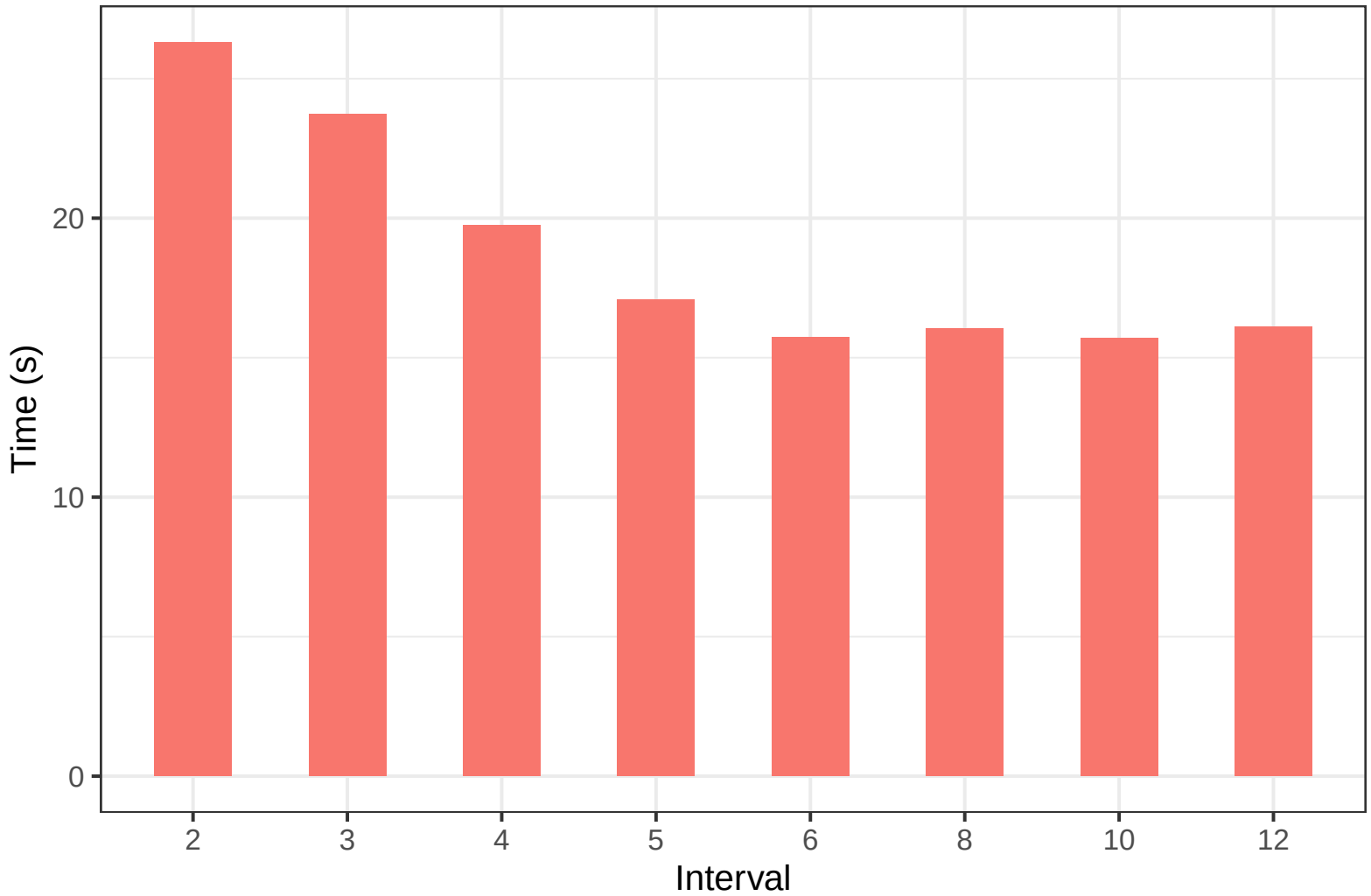
- Relative performance of Phase 1 using synthetic datasets.



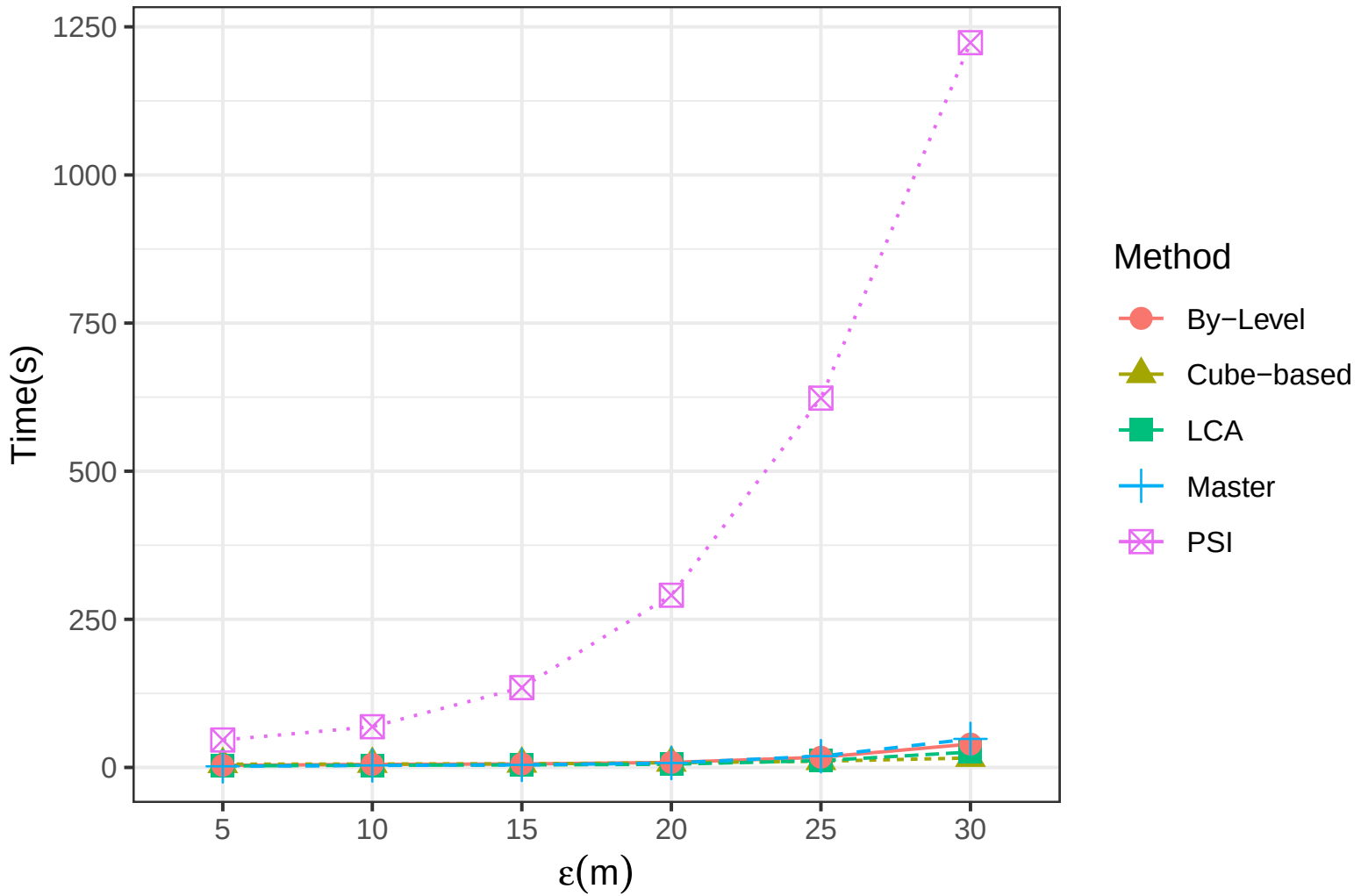
- Finding best *step* value for By-Level alternative.



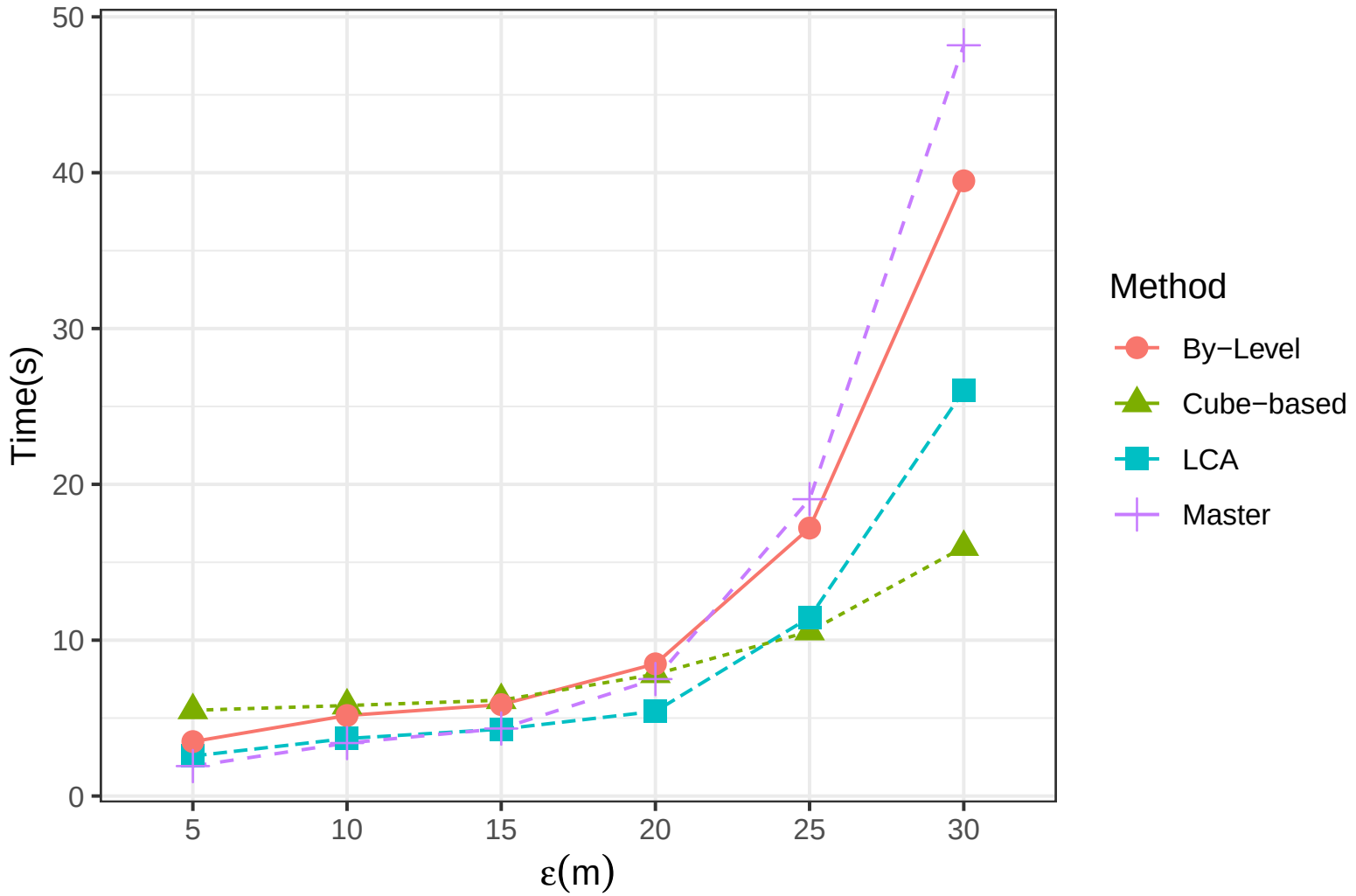
- Finding best *interval* value for Cube-based alternative.



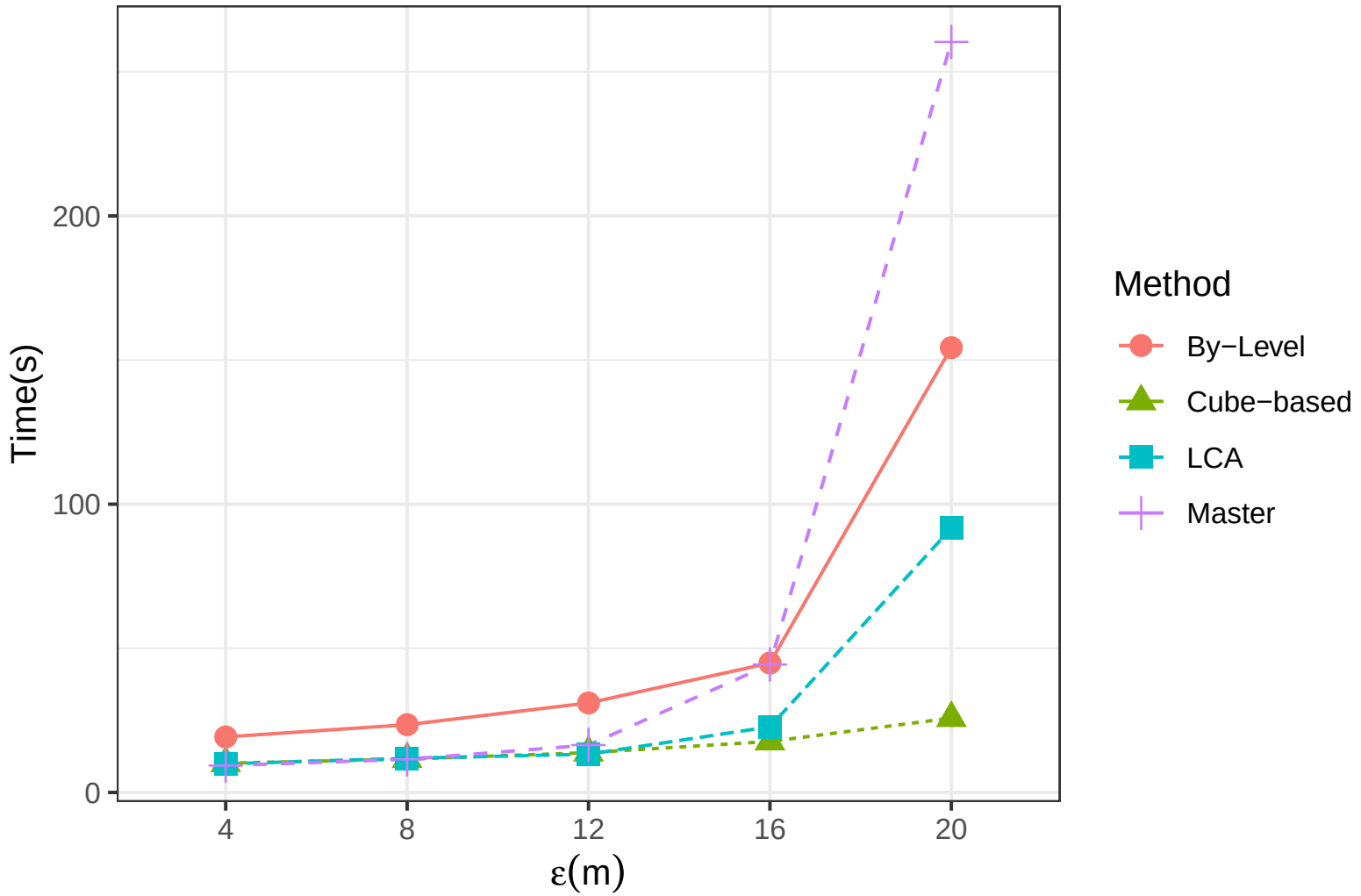
- Scalable alternatives vs standard PSI.



- Scalable alternatives in LA25K dataset.



- Scalable alternatives in LA50K dataset.



Thank you!