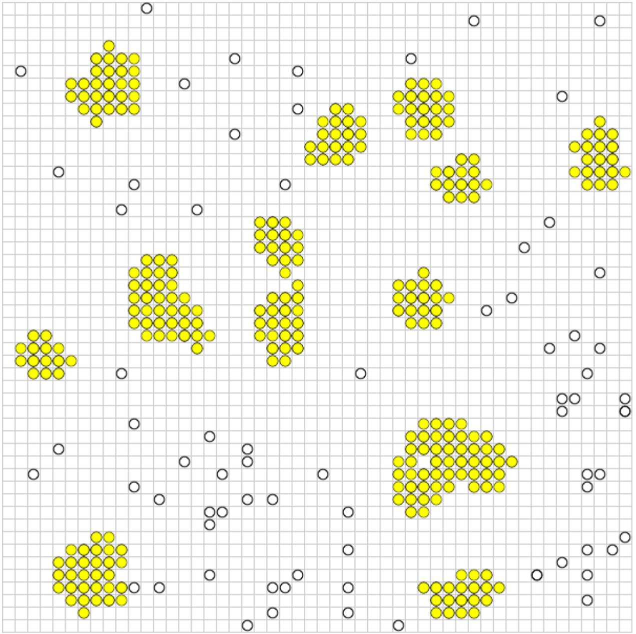
Quadots

C++ library for simulating the behavior of points in a 2d space.

Simulations

- Physics
  - Flock behavior
  - Collision
- Conway's Game of Life
- AI simulations
- Data Analysis
- etc



Use

Tools

Simulator

Renderer

Control

Basic element types

- Points(x, y)
- Dots(x, y, dir, vel)
- MyElement(x, y, hairstyle)

Use

Create behaviors

Create elements

Create simulation

Pass behaviors and elements to sim

Run simulation

Get final state

Elements

```
Point
{
   Update()        // does nothing
   get_x()
   get_y()
   bindex
}
```

Dot

Dot

- + velocity
- + angle (direction)

Update function changes x/y based on velocity and angle.

Rule

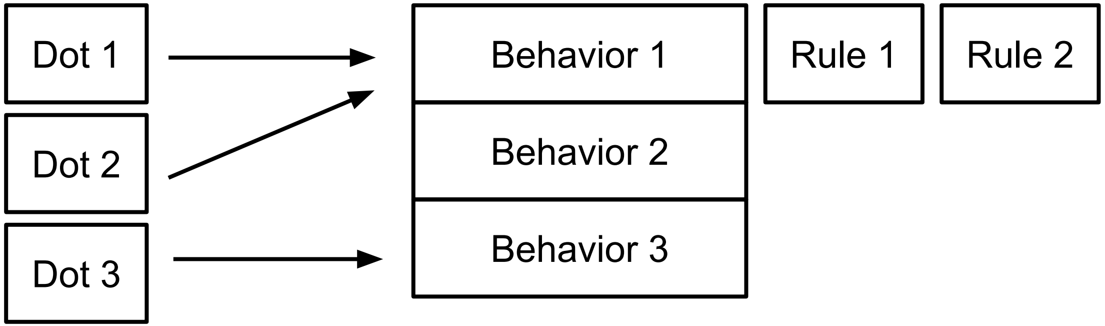
Rule: &function (element_p, control&)

```
conform(Dot_p  d, Control& c) {
    a = c.get_avg_dir();
    d->set_dir(a);
}
```

Behaviors

Behavior: { rule1, rule2, rule3 }

Assign an element a behavior

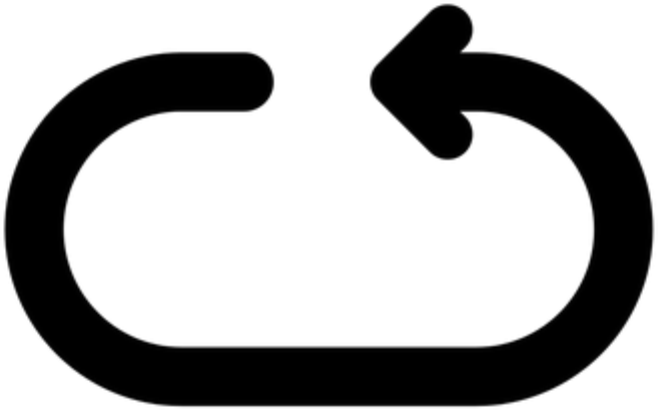


Step

Simulator

For each element:

- Get behavior
- Apply all rules
- Update()
- Push to new state



Process

```
Simulation *sim = new Simulation(800, 800);

int b = sim->CreateBehavior(rotate);

Dot d = Dot(100, 100, 0, 1, b);

sim->CreateElement(d);
```

Run

```
sim -> Run ( step count , print )
```

or

```
sim -> Run ( step count , renderer )
```

Abstraction

Essentially the purpose of the library.

- No need to worry about simulating.
- No need to manage allocation.

Flexibility

All classes are templated to fit element type

Custom:

- Elements
- Control functions
- Renderer

Application

Boids

Flocking behavior of birds

Separation

Steer to avoid crowding local flockmates

Alignment

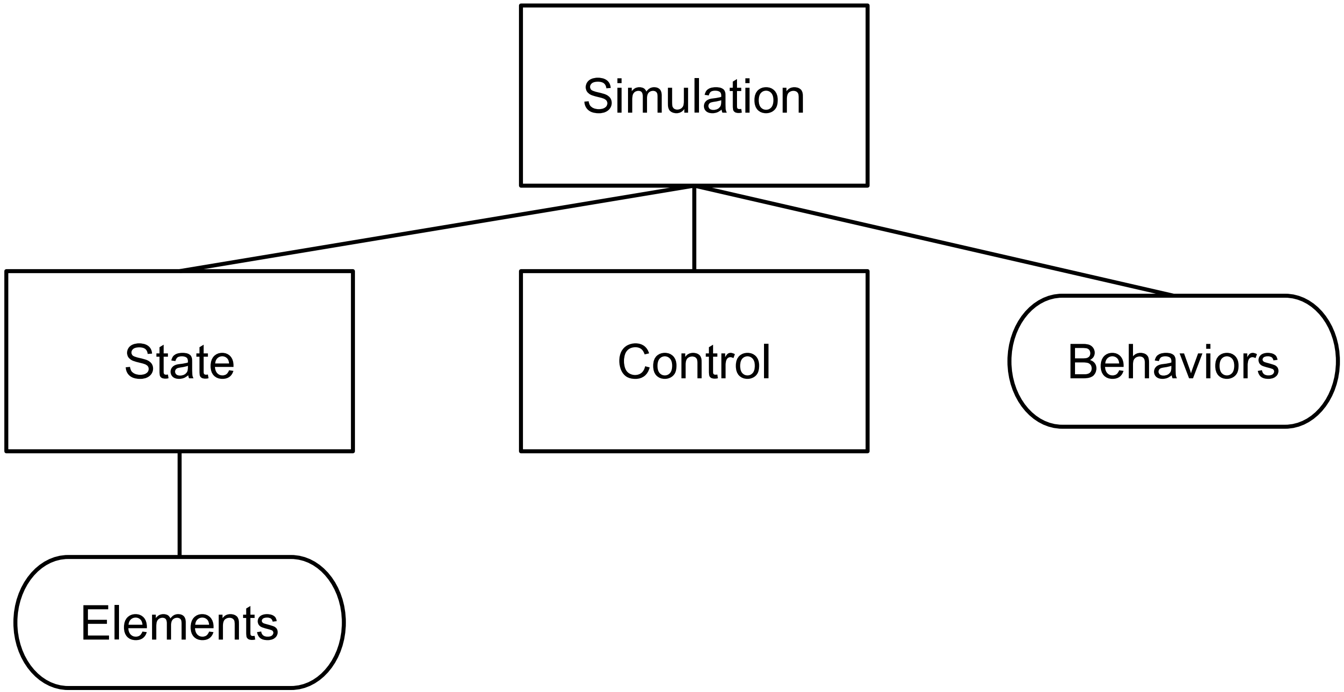
Steer towards the average heading of local flockmates

Cohesion

Steer to move toward the average position of local flockmates

Implementation

Classes



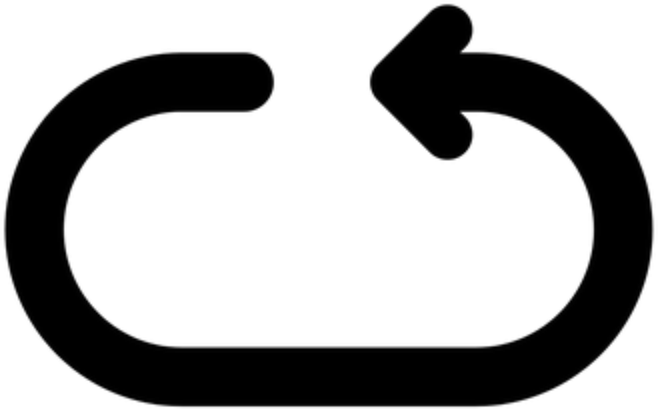
Process

Get elements from state

For each element:

- Create copy
- Get behavior using index
- Call each rule (pass control)
- Add copy to new State

Replace old state with new



Control

get_distance(a,b)

avg_x()

avg_x({elements})

neighbors(a, radius)

Quadtree

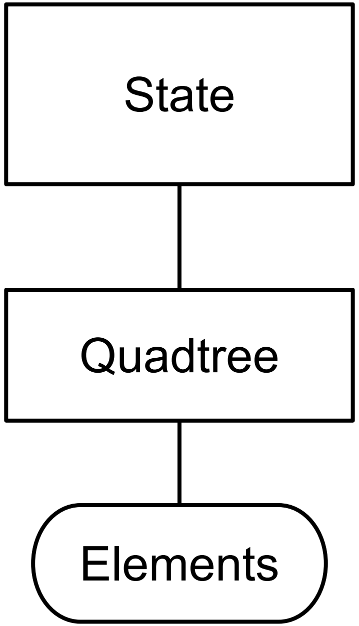
Quadtree

Element storage

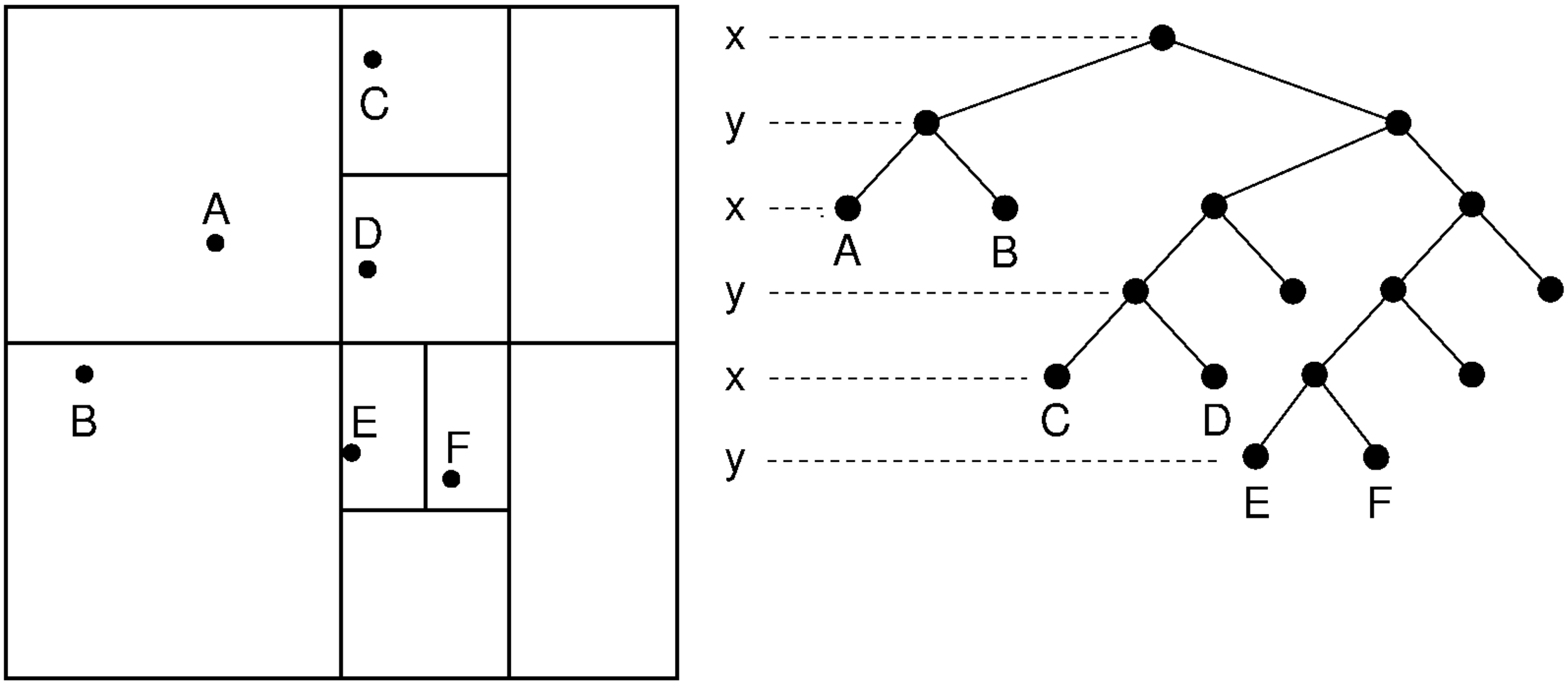
Great spacial structure

Elements in range

- Neighbors
- Collision



Quadtree



(a) (b)

Nearest Neighbors

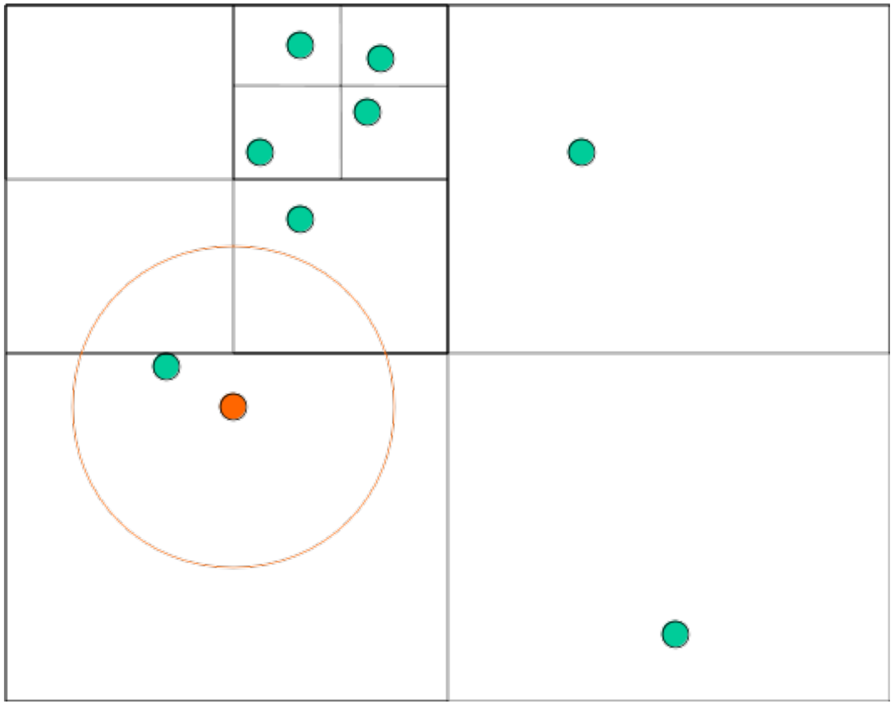
Near neighbor (range search):

- put the root on the stack
- repeat
  - pop the next node T from the stack
  - for each child C of T:
    - if C intersects with the ball of the radius

      if C is a leaf, examine point(s) in C

      else, add C to the stack

Nearest Neighbor



Much less points to calculate distance between.

Measurements

Measurements

Focus: getNearestNeighbours() function

We measure performance of Quad trees vs. Brute Force

Terminology: 1. MO = MAX_OBJECTS

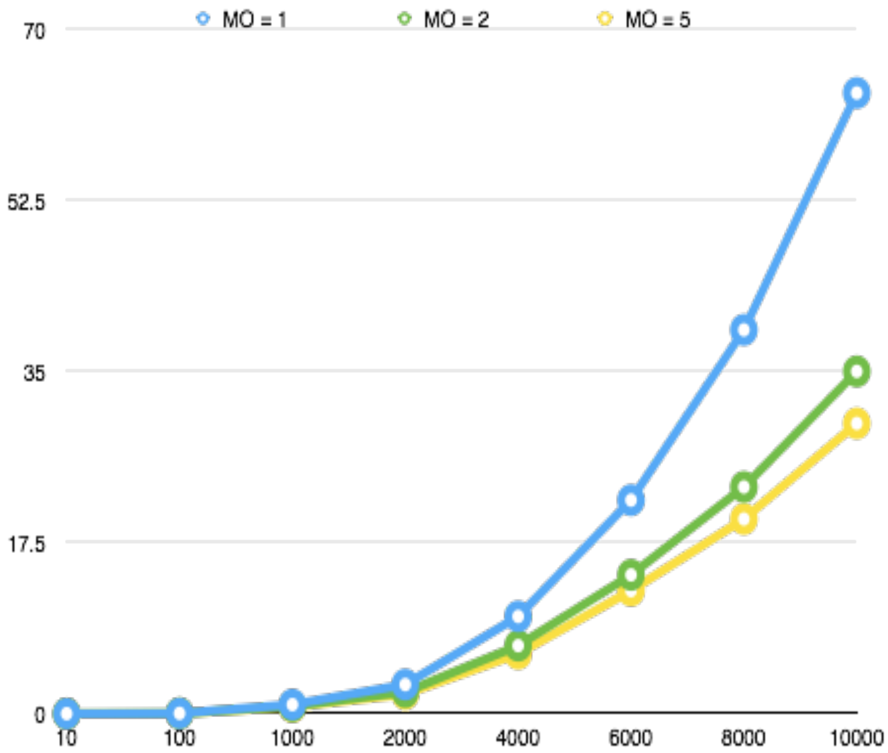
2. Density = No. of points in sim.

Measurements

1. MO = {1,2,5}

Conclusion:

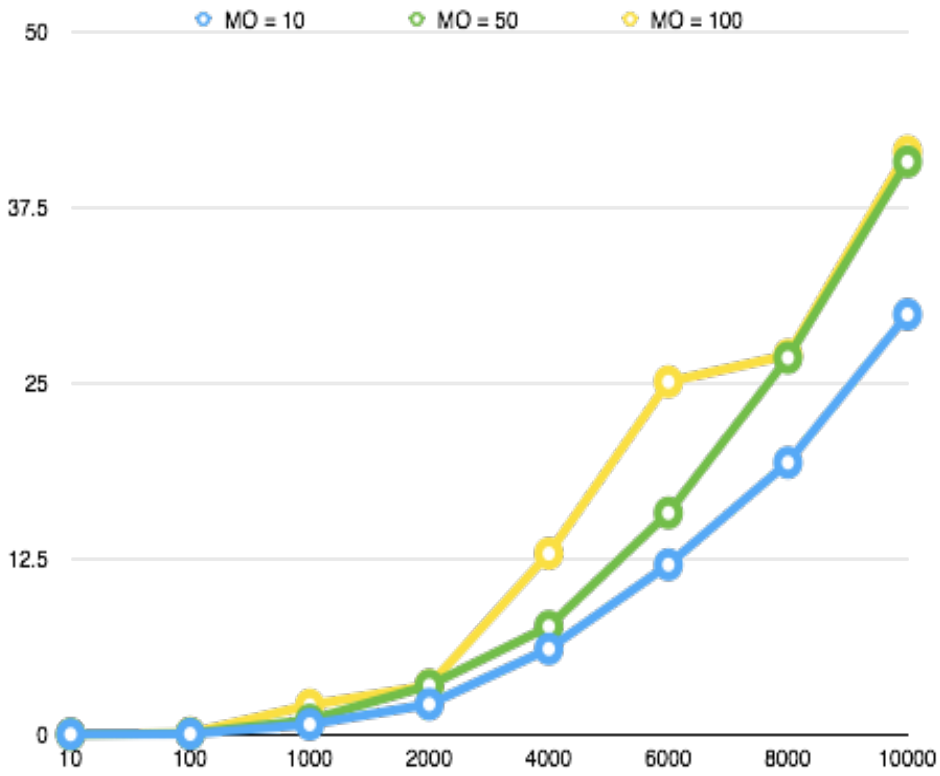
MO = {1} is bad.



Measurements

2. MO = {10,50,100}

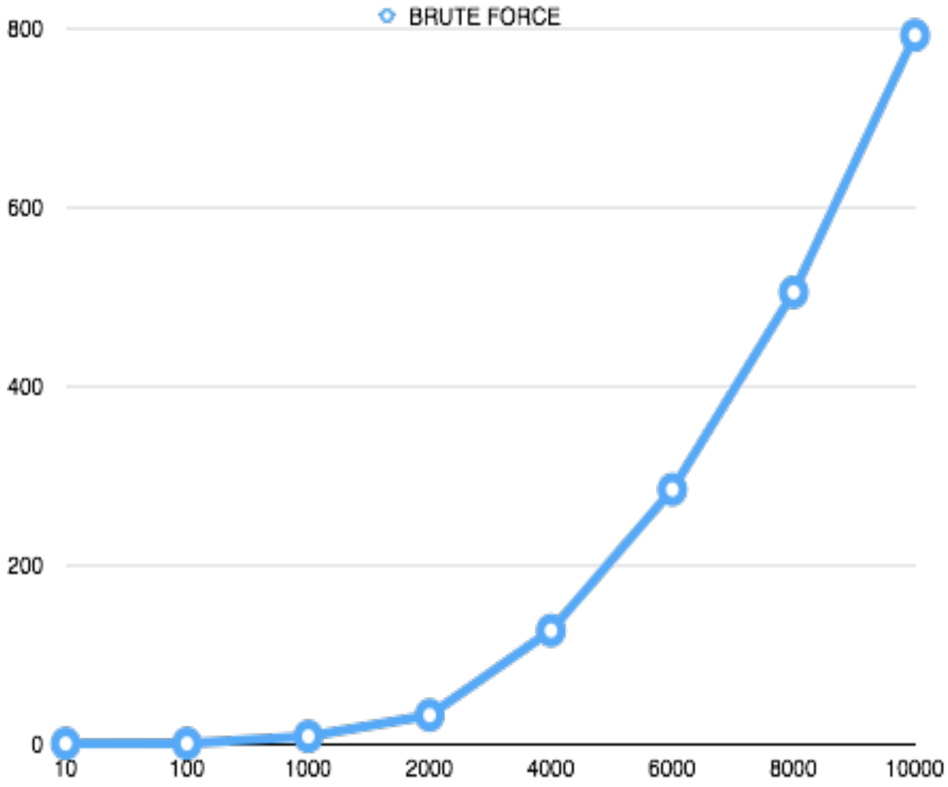
Conclusion:
As MO increases, quad trees reduce to brute force.



Measurements

3. Brute Force

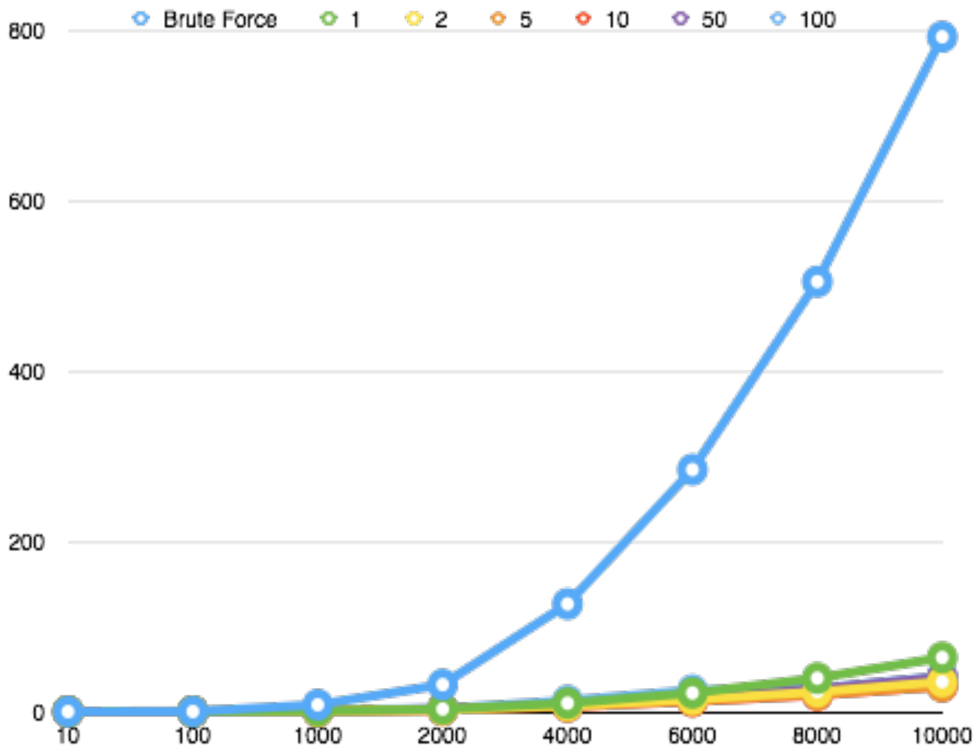
Conclusion:
Time taken by BF increases exponentially.



Measurements

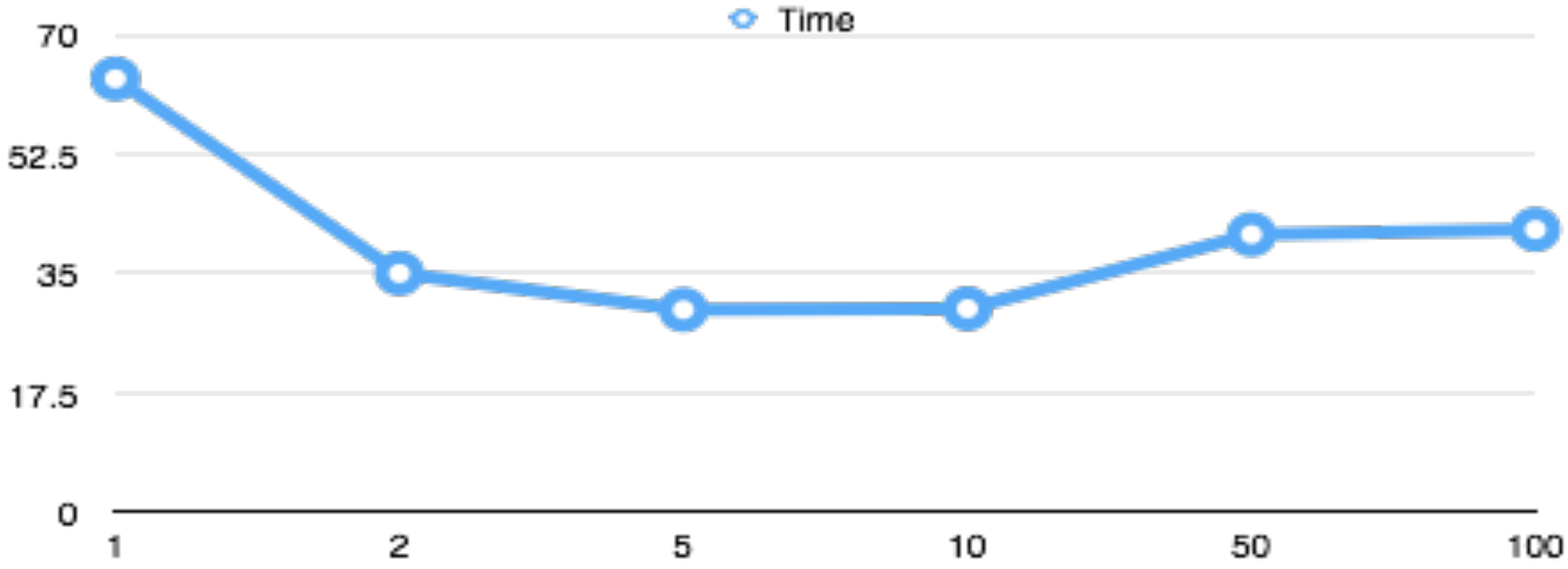
4. Quadtrees vs. BF

Conclusion:
Quadtrees are 25x faster at high densities.



Measurements

5. Optimal MO???



Acknowledgements

Thanks Professor!

Thanks David!

Fin

Team

Souren Papazian
Surashree Kulkarni
Srilakshmi Chintala

Code

https://github.com/SourenP/quadots