Verification of Moving K-Nearest-Neighbor Query

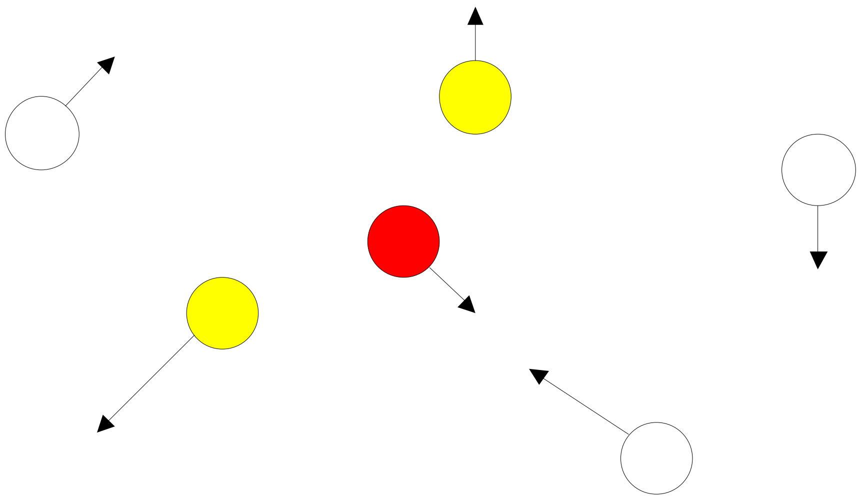
Marcin Kwietniewski

Agenda

- Problem revision
- Implementation overview
- Goals
- JPF issues
- Model modifications
- Future work

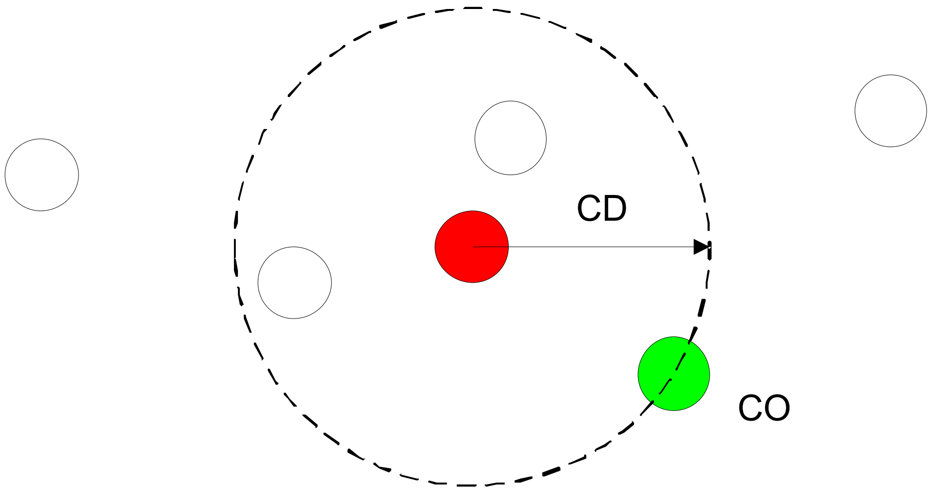
Problem Revision

- A mobile network: server, base stations, mobiles
- We are interested in k nearest neighbours of some mobile
- Mobiles are constantly moving

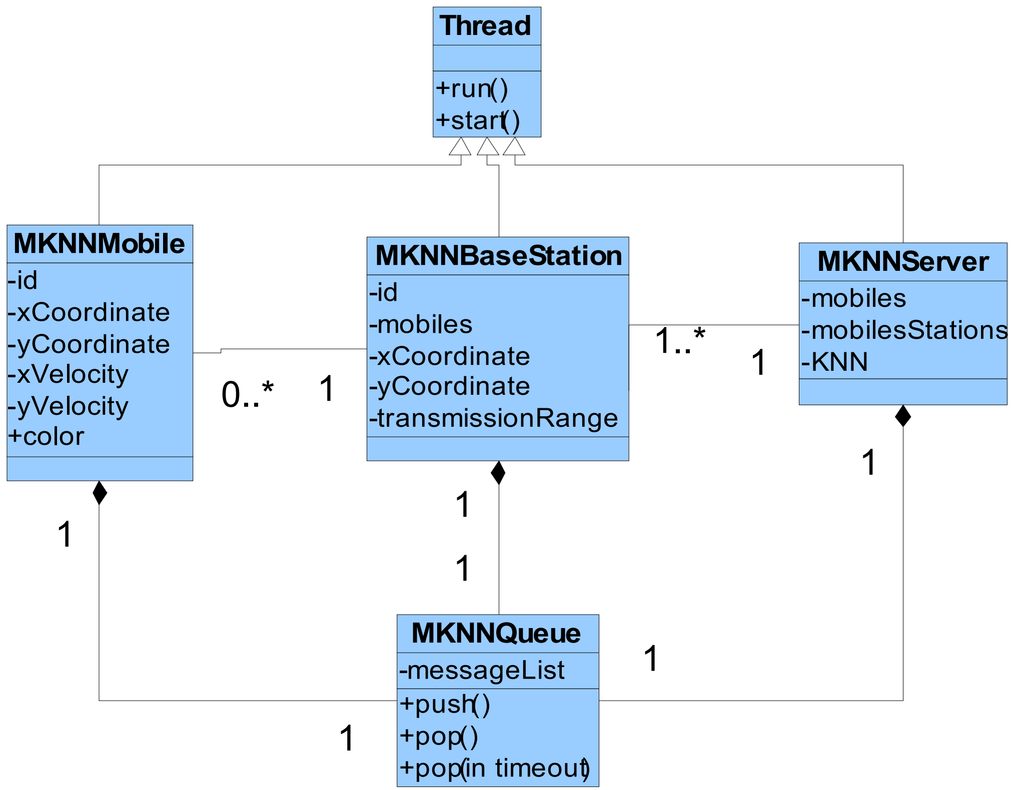


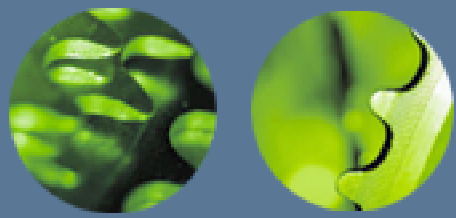
The Algorithm

- First phase: we compute the initial result by asking the mobiles about their positions
- Continuous processing: we keep track of positions of the query owner and the critical object



Implementation overview





Message passing

- Mobiles report their base station changes
- Initial stage:
  - position request broadcast from the server
  - mobiles report their positions
- Continuous phase:
  - query owner's position and critical object's position are broadcasted
  - mobiles report changes in the query result

Goals

- What to verify?
  - Simulation doesn't crash
  - Query gives the right answer
    - K doesn't change
    - No more than one object should be in the result and is not
- In what situations?
  - At least 2 base stations
  - At least 4 mobiles: owner, critical and 2 mobiles simultaneously changing the result

JPF issues - randomization

- Positions of mobiles random – initially in semi-continuous domain (double)
- JPF branches for every possible random value
- Floating point values can be handled using DoubleThresholdGenerator heuristic
- Solution:
  - Limited starting positions
  - Limited possible velocity values

JPF issues

- Thread termination:
  - Initially, the simulation didn't stop at all
  - Process termination causes JPF to crash (*join()* problem?)
  - Solution:
    - MKNNLauncher waits on a semaphore for all the mobiles to finish.
    - Mobiles have a limited number of „life cycles”

JPF issues

- *sleep(milliseconds)* method has no effect
- On the other hand, JVM is not a real-time system, there are no guarantees on timing
- First bugs found:
  - Query owner has to start the query after he logs into the network
  - Launcher has to log in all mobiles automatically

State space problems

- Initial tests with only 4 mobiles and 1 base station
- JPF runs out of memory (1GB) after ~100,000 states checked (quickly)
- Ensure that partial order reduction (POR) didn't miss anything:

Server.run():

messageQueue.pop()
Verify.beginAtomic();
all the processing
Verify.endAtomic();

BaseStation.run():

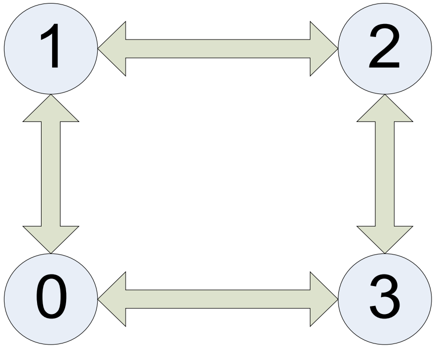
messageQueue.pop();
Verify.beginAtomic();
pass the message further
Verify.endAtomic();

State space problems

- No big improvement observed
- # mobile positions allowed too big:
  - in 3 steps with 1 velocity value allowed 49 positions van be reached
  - Execution paths = $49^{\#mobiles}$ * #initial configurations*message queue states
  - We can't rely on state comparisons

Model change

- Only 4 mobile positions allowed:



- Mobiles jump counterclockwise/clockwise or don't move (3 choices)
- Great reduction in state space
- Hopefully, all interesting events from the original model will happen here

Results

- JPF wasn't able to check the whole state space
- ~50,000,000 states checked
- BFS loses most of its time on all possible initial configurations, therefore DFS is more interesting
- I prefer to check a more complex model to some extent, than a simple model completely

Race conditions

- No race conditions found!
- That might be due to properly synchronized MKNNMessageQueue
- Obviously, message order depends strongly on scheduling

Future work

- Focus on validating the query result
- I might work on the allowed delays in terms of mobile steps
- Validate a two-state model?
- Validate a mobile permutation model?

The end

Questions?