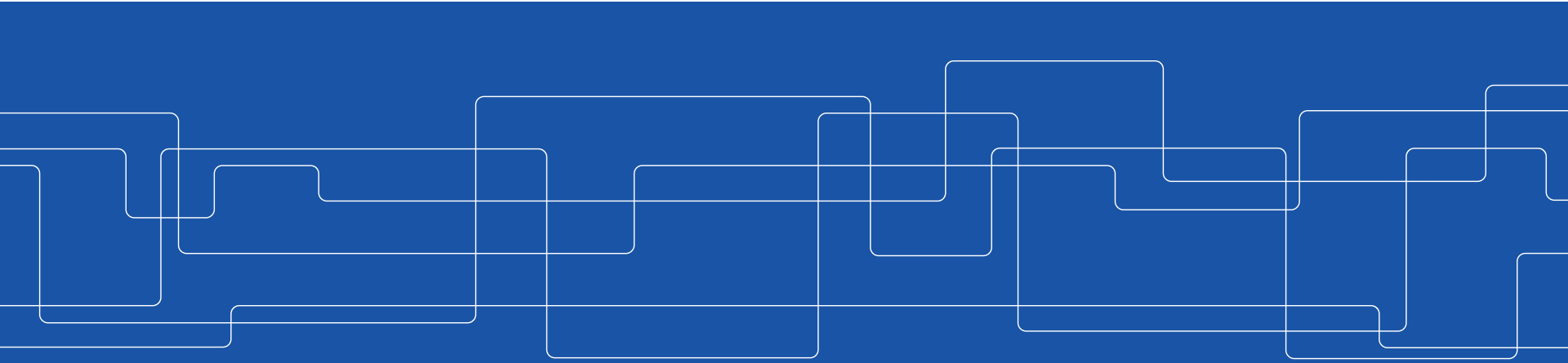
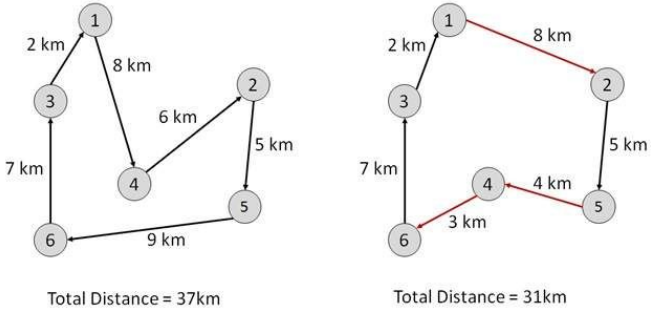
Team 10: Taking out the trash

Alex Sundström	alexsu@kth.se
Konrad Magnusson	konradma@kth.se
Tim Olsson	tiol@kth.se
Henrik Karlsson	henrik10@kth.se



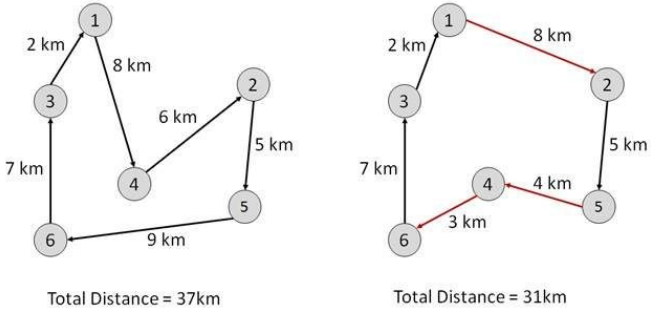
Project presentation in Artificial Intelligence (DD2380)

Travelling Salesman Problem (TSP)



- Travel to all cities in the shortest time/distance possible
- Find the shortest hamiltonian path, i.e. shortest path that covers all vertices.

Travelling Salesman Problem (TSP)



- Travel to all cities in the shortest time/distance possible
- Find the shortest hamiltonian path, i.e. shortest path that covers all vertices.
- Complete search in O(n!)

Multi-Agent TSP

- Visit all cities in the shortest time possible
- All salesmen travel equally fast

Multi-Agent TSP

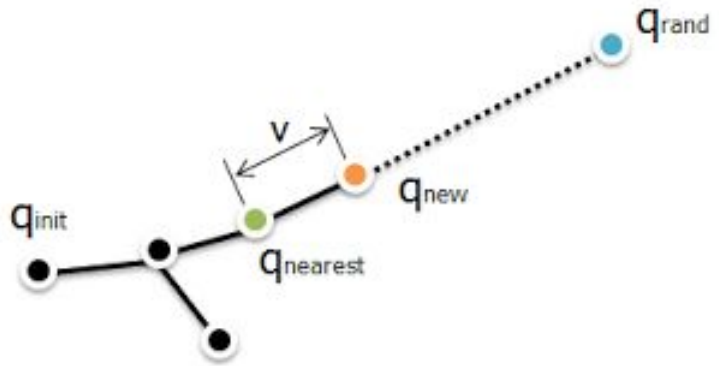
- Visit all cities in the shortest time possible
- All salesmen travel equally fast

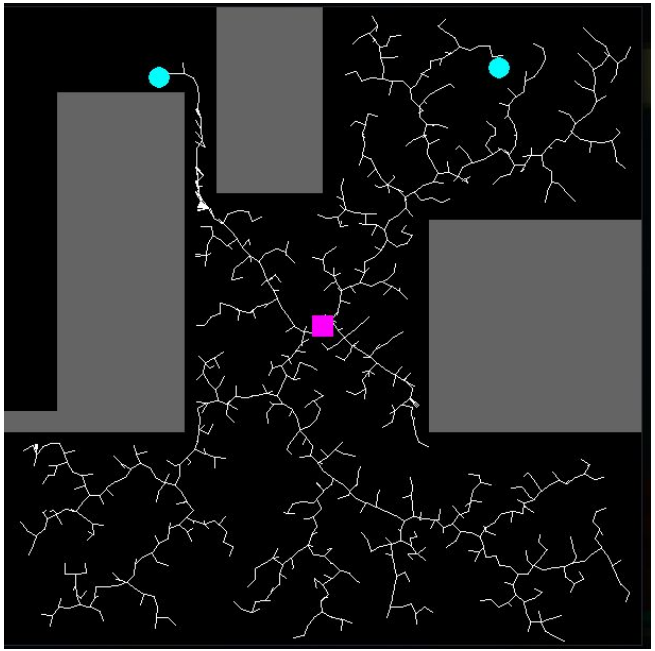
→ The last salesman needs to finish as soon as possible

Solve in two steps:

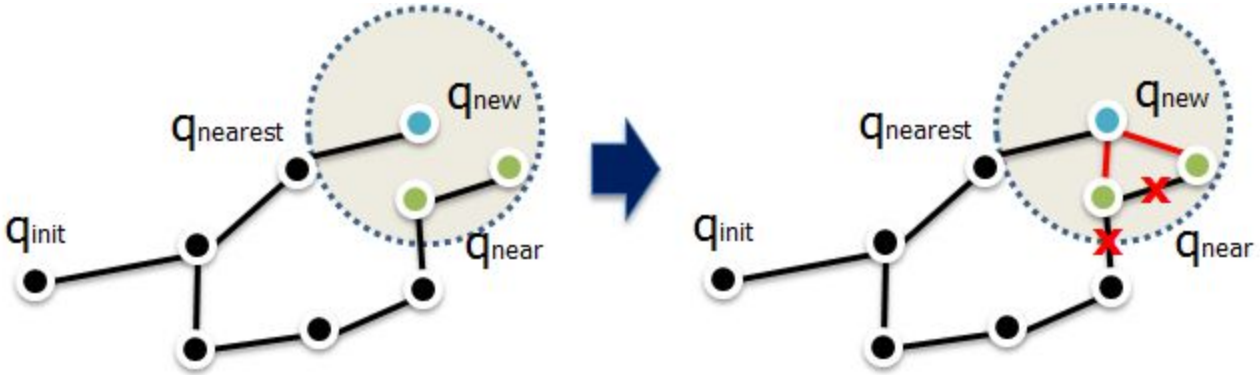
1. Construct the graph
2. Find solution by solving MA-TSP

RRT



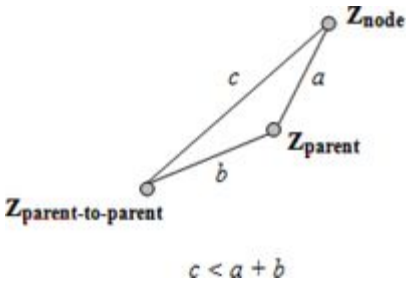


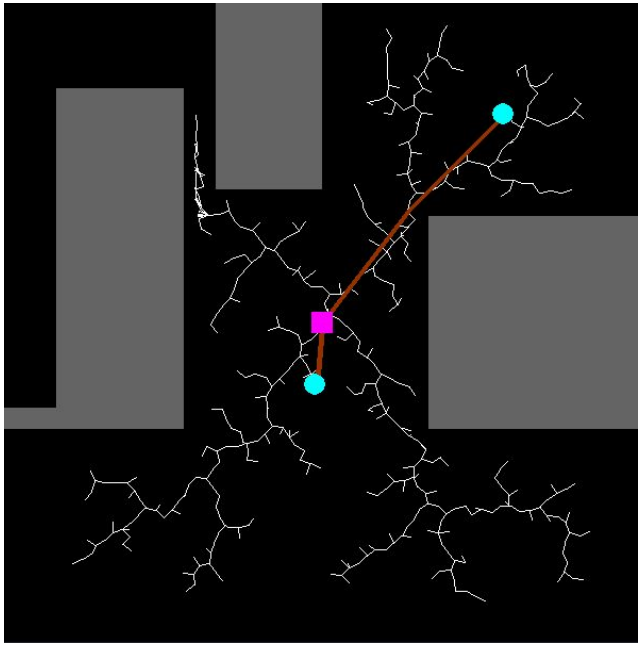
RRT*



Path optimization

- Triangle inequality





PDDL

- PyDDL, a PDDL planner in Python by Gary Doran[1]
  - Can use an A* algorithm
- We let the heuristics = 0, we got Djikstra's algorithm
- Works fine when # trash cans < 10

[1]GitHub repository: https://github.com/garydoranjr/pyddl

PDDL - Domain

```
Objects:
     0, …, N-1   agent                    -- N: number of agents
     0, …, N+M-1 trash_can                -- N + M: agents + trash_cans
```

PDDL - Domain

```
Objects:
     0, …, N-1   agent                      -- N: number of agents
     0, …, N+M-1 trash_can                  -- N + M: agents + trash_cans
Init:
     at(0,0), …, at(N-1,N-1)                -- Agents at "Trash cans" 0,2, …, N-1
     unchecked(N), …, unchecked(N+M-1)      -- Real trash cans N, N+1, …, N+M-1
```

PDDL - Domain

```
Objects:
     0, …, N-1  agent                    -- N: number of agents
     0, …, N+M-1 trash_can               -- N + M: agents + trash_cans = locations
Init:
     at(0,0), …, at(N-1,N-1)             -- Agents at “Trash cans” 1,2, …, N
     unchecked(N), …, unchecked(N+M-1)   -- REAL trash cans N, N+1, …, N+M-1
Goal:
     checked(N), …, checked(N+M-1)       -- Check all REAL trash cans
```

PDDL - Actions

```
Action: Check
   Parameters:
        A1 agent                    -- the robots
        T1, T2 trash_can            -- locations/trash cans
```

PDDL - Actions

```
Action: Check
   Parameters:
        A1 agent                    -- the robots
        T1, T2 trash_can            -- locations/trash cans
   Precondition:
        at(A1, T1)
        unchecked(T2)               -- don't check twice
```

PDDL - Actions

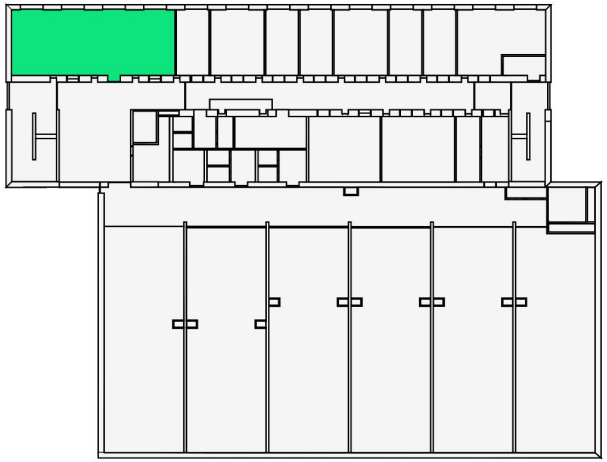
```
Action: Check
   Parameters:
        A1 agent                    -- the robots
        T1, T2 trash_can            -- locations/trash cans
   Precondition:
        at(A1, T1)
        unchecked(T2)               -- don't check twice
   Effects:
        travelled(A1, T1, T2)       -- keep track of the distance travelled
        ~at(A1, T1)                 -- leave T1
        at(A1, T2)                       -- move to T2
        checked(T2)                 -- T2 is checked
        ~unchecked(T2)
```

PDDL - cost function

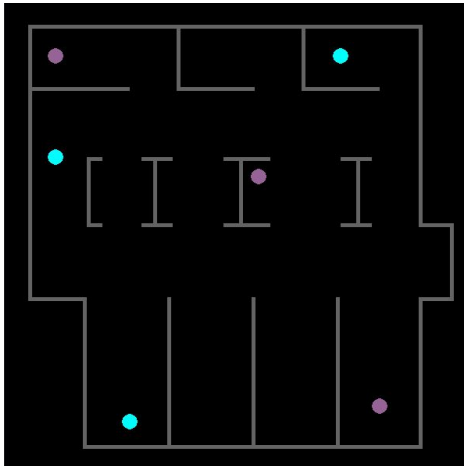
- Cost = $\max_{i \in A}(\text{dist}(a_i))$
  - where $\text{dist}(a_i)$ is the distance a_i has travelled.
- Minimizes the time to check trash cans
  - Doesn’t minimize the total distance travelled
- We get the distance from RRT* and travelled/3 predicate

Test cases

- The suggested test case was Teknikringen 14
- We experimented with some other cases as well



Teknikringen 14

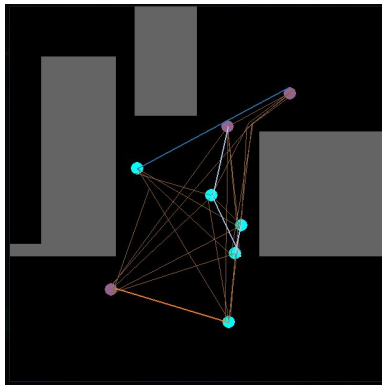


Our “artistic” interpretation

Custom test cases

- Can define agents, targets and obstacles in text files
- Was easy and fast to implement

```
r 250 350
t 380 420
t 25 500
t 500 100
t 322 532
t 500 450
w 50 80 120 320
w 0 380 50 20
```



Results

- RRT makes up the bulk of the execution time
  - The RRT phase is not deterministic, can results in different plans on same test case
- The planner is pretty fast though for small N
  - However, the time complexity seems to be ~O(N!)
  - Bad scaling

Future work

- Faster pathfinding
  - RRT connect
- More iterations of RRT to reduce variance in the distance matrix
  - Simulate the robot's path better
- Better planning
  - Use heuristics for PDDL if N is big
  - Just use A*

Thank you for your time!

:^)