

UTEK2022

Programming

Team 2

Project Summary: A Path-Planning Algorithm for
Cleaner Robots based on Greedy Heuristics

January 15, 2022



Problem Statement

- Energy crisis needs immediate attention
 - Aim: to conserve energy
- How?
 - By reducing the energy consumption of cleaner robots
- Central Problem
 - To skillfully allocate resources according to specifications of each available cleaner robot (movement speed and cleaning speed)

Problem: Key Aspects

Travelling Salesman Problem (TSP)

- Optimize the path of cleaner robots to locations which require sanitization
 - Reduce repetitive routing is an effective means to reduce energy consumption
- The solution to this problem is an appropriate planning of robot routes, such that the cleaning requirement is met with the lowest energy cost

Approach: Heuristics

- The TSP is a NP-hard problem
- There is no *optimal* solution to the TSP
- Instead of using a brute-force approach, heuristic algorithms with higher efficiencies and lower resource requirements are considered

Greedy Algorithm

- The greedy algorithm makes “the optimal choice at each step in order to find the overall optimal way to solve the entire problem.”
- It balances simplicity and effectiveness.

Assumptions

The robot will always move diagonally first and then horizontally or vertically.

- Justification: This assumption saves computational resources to include redundant if-statements.

The robot will only detour when an obstacle is blocking its move to the intended next step.

- Justification: This assumption saves computational resources to calculate (e.g. using depth-first search) the optimal path between nodes in the presence of obstacles

The robot will always target the closest spot regardless of obstacles potentially in its way.

- Justification: This assumption saves computational resources to calculate (e.g. using Dijkstra's algorithm) the shortest path between nodes in the presence of obstacles

Sample Implementation


```
def getMatrix(x,y):  
    res = []  
    for i in range(len(x)):  
        not_rly_res = []  
        for j in range(len(y)):  
            inter = (min(abs(x[j]-x[i]),abs(y[j]-y[i]))+\  
                abs(abs(x[j]-x[i])-abs(y[j]-y[i])))  
            not_rly_res.append(inter)  
        res.append(not_rly_res)  
    return res
```


- We determine that the shortest “robot distance” (a combination of Euclidian and Manhattan distances) between two spots is achieved by moving diagonally for as much as possible. It is based on this conclusion that we optimize the robots' paths.


Program Evaluation


Advantages



 Relatively low computational complexity (both time and space)


 Straightforward: easy to understand as a reader and implement as a programmer


 Widely generalizable: can be adapted to work in environments with unknown obstacles

 Practical: matches effective working procedures by current cleaner robots


Disadvantages



 Less competitive runtime: this program is written in Python, a language that generally requires higher runtime

 Inaccurate heuristics: the Greedy Algorithm is one of many heuristics; other algorithms might deliver better performances

 Not optimized: excessive use of if-statements and inefficient use of loops

 Incomplete: Due to time constraints, some parts of the program is incomplete (they are, however, completed with pseudocode)

Next Steps: Optimization

- Complete the remaining portions of the program
- Consider various test cases (including boundary and unexpected inputs)
- Explore better algorithms (e.g. genetic, k-optimal, Lin-Kernighan, etc.)
- Convert redundant functions in the five different parts into generalized helper functions
- Utilize external libraries to simplify tasks and achieve higher efficiency

Thank you!

Contact

Programming Team 2

Carl Ka To Ma, carlk.ma@mail.utoronto.ca

Zihuan Jiang

Shenxiao Zhu Xu

GitHub: <https://github.com/macarlo8/utek2022>