Pick-up and delivery using MAPF

Seminar on AI 2018

Team #1

David Nohejl

Chaman Shafiq

Věra Škopková

Introduction

OUR TOPIC

Problem definition

- Given:
  - Map of the environment
  - Pick-up locations and goal locations
  - Tasks
- Multiple robots – every robot should complete a task
  - 1) pick up an item at a pick-up location
  - 2) deliver that item to goal location as quickly as possible

Plans for individual robots

- Offline planning → execution on robots
- Abstraction – discrete steps
- Plans should be collision free
- Constraints
  - Two agents can not be at the same place at the same time
  - Only one agent can go through one way at one time step

Tasks

- To study and implement some effective algorithm for MAPF and to use it in our problem
- To observe how is the plan executed in ozobots and:
  - To try to solve problems that appear because of inaccurate start of execution of the plans
  - To try to react to obstacles that appear in the map

Supporting programs

GRID DESIGNER
OZOCODE GENERATOR

Grid Designer

- Demo

Ozocode Generator

- Written in C#
- Generates xml code that is possible to store into ozobot
- Possible to open generated code in ozoblockly editor
- Supports only basic commands we need in our project
  - Follow line, turn left, turn right, go forward, go backward, wait, stop motors
  - Say direction, set top light color, turn top light off

Ozocode Generator

- Input
  - Text file with sequence of „turns“ – left, right, forward, backward and wait
  - Describtion of the path of one ozobot
- Output
  - File with ozocode for going exactly the directions from input file

Algorithm and results

WHAT WE HAVE DONE

MAPF algo - Conflict Based Search

Algorithm 1: high-level of CBS

Input: MAPF instance

$R.constraints = \emptyset$

$R.solution$ = find individual paths using the low-level()

$R.cost = SIC(R.solution)$

insert R to OPEN

while OPEN *not empty* **do**

  P ← best node from OPEN // *lowest solution cost*

  Validate the paths in P until a conflict occurs.

  if *P has no conflict* **then**

    return P.solution // *P is goal*

  C ← first conflict (a_i, a_j, v, t) in P

  foreach *agent* a_i *in C* **do**

    A ← new node

    A.constriants ← P.constriants + (a_i, s, t)

    A.solution ← P.solution.

    Update A.solution by invoking low-level(a_i)

    $A.cost = SIC(A.solution)$

    Insert A to OPEN

CBS - kinda

- Quick & dirty implementation of conflict based search
  - Uses BFS instead of A*
  - Branches for all optimal paths
  - “Swap” problem is solved somewhat arbitrarily
  - Slow, but seems to be working ☺
- Input
  - Compatible with the output of the Grid Designer
- Output
  - Compatible with the input of the Ozocode Generator

Running on ozobots - observation

- Good map quality, appropriate thickness of the line
  - Otherwise ozobot can miss junction or it can detect junction even on the straight line
- Equal distances between nodes
- Enough space between nodes (to avoid crashes of ozobots on neighbouring nodes)
- Run multiple ozobots in the same moment (sometimes ozobot does not find the line on the beginning and the whole experiment has to be repeated)
- Waiting time (when no move is done) depends on the length of the line between the nodes
- Video

What next?

HOW TO CONTINUE

Future work

- To implement more efficient algorithm
- Kinetic constraints
- To finish Pick-up and Delivery
- To react to obstacles in the map that were not present when searching the path
- To create simulation of ozobots in the Grid Designer
  - To be able to try with more agents than we have
- To add new functions to Ozocode Generator
  - Loops for obstacles detection
  - Maybe to be able to do an undo step