Implementation documentation

petr.smid

March 2023

1 Introduction

The mapd visualisation is WPF application created using c in .NET 6. Architectural model is MVVM . The supported OS is windows, for implementation reasons, wpf model is crucial for visualisaion. Application is public usable for research or private purposes. No login is required, no security is implemented, recommended for local usage only. Please see user-documentation for detailed explanation on app behaviour and functions.

2 Main Components

2.1 Main window

The window consists of grid with 3 columns, separating main settings sections; input, problem settings, scenario settings.
The core of the application. Takes input, validates and loads data into mock backend database. Loads setting chosen by user, fires presolve event (more in Routing component) and enables user to modify database data for current problem by adding agents/orders or clearing scenarios or entire loaded map.
Data, map or scenario can be reloaded and contents of database updated. Upon closing, application stops.

2.2 Simulation window

This window is a grid, scaled to size of each monitor. Number of rows and collumns exactly corresponds to the input map dimensions.
Simulation is conducted by coloring each tile with color of corresponding agent/order in every step of the plan. Tiles recolored in given frames correspond exactly to one step in plan.
Currently active orders are highlighted with the same color as the correspondent agent, rest is grey for better overview. Background thread resets every 100ms checking for pause button press. After executing one time step of the plan, the window is frozen for 50ms. If needed, the simulation can be easily slowed down or sped up.

Plan validation triggers fullscreen mode of the window, draws grid such that intersections correspond to the tiles (and locations). The reason is better orientation in overall validation. Plan validator finds biggest possible non-conflict path segment and recolors its edges. Freezes for 100ms and the application takes screenshot and saves it. At the end of the process, it takes all pictures and creates Plan validation pdf using iTextSharp.

2.3 Plan components

- PlanCreator
  When user proceeds to solve given task, planCreator loads corresponding numbebr of orders from database, obtain and set solution settings from database, creates problem objects, runs given algorithm and outplut Plan instance

- PlanValidator
  Receives plan instance, extracts biggest possible non collision segments using bfs, and sends them for visualisation and screenshot to the simulation window.
  After all screenshots are created, exports pdf containing visualised non collision paths, time and agent info.

- ColorAssigner
  Receives problem definition, runs MTSP and according its results, returns assignment of orders to each agent (and thei colring)

2.4

Routing component Non essential component for order assignment. The application uses OR-TOOLS library and its routing components. Based on problem definition, Routing problem is modeled and solved.
The routing problem consists of depots, locations, distance matrix, time, capacity and order constraints. Time windows can be used for better results in known environment, however they are not suitable for online environment.
Results are only approximate and do not lead to optimal solution, that is not the purpose of this application. User is more than welcome to create his/hers own order assign modul.

3 IO sections

3.1 Plan Reader

Plan reader receives path to .plan file, checks validity of header and start executing steps line by line. If at any point invalid plan step is found, warning is triggered, visualisation is stopped and reset to initial position.

Plan is executed as follows: All plan steps for given time are loaded into memory. All previous positions of agents and orders are blended (if not occupied with anothe agent or unfinished order) and new tiles are colored according to the plan scheme. Dispatcher is used to forward the changes in real time onto the visualisation window.
Reading entire file into memory can cause poblems for large scale planns and thats why the Plan reader component avoids it.

3.2 Plan Writer

Plan writer receives instance of Plan class and path to file (usually mapName.plan). Writes header consisting of info about map, agents, orders and assignment of orders to the agents.
Instances of PlanSteps are then written on each line for every plan step. For example, please see example.plan or user-documentation.

3.3 Map Parser

Receives path to .map file. First checks validity of header and start loading tiles line by line. If at any point invalid line is found, warning is triggered, loading is stopped and completely reset.
Map parser creates matrix of position of given dimensions in header and loads in into database. Locations have two types: free and wall.

3.4 Scenario Parser

Every scenario is created for specific map. If scenario map name doesn't correspond to the loaded map name, warning is triggered.
Receives path to .scen file. First checks validity of header and and start loading orders and agents line by line. If at any point invalid line is found, warning is triggered, loading is stopped and completely reset.
Scenario parser creates list of orders and agents with their complete info (locations, ids, etc) and loads it into database.

4 Algorithms

The application provides multiple algorithms for MAPD insance solutions: TP, TPSP-dh, Central-A*, Central-CBS and many subalgorithms as general TSP, greedy or Hungarian for task assignment.
The algorithm component is completely separated from other logic involved in the application. Every algorithm is given so called problem object, containing instance of a MAPD problem and output solution plan (empty if no solution is found).

4.1 Problem object

It is instance of an MAPD problem. It must contain locations, agents and orders. Upon selecting number of orders by user, the agents and corresponding number of loaded order objects are copied using their :ICloneable interface. The locations do not need to be copied, they are not modified within the algorithm except marking occupied states. Those are reset every run anyways.
Problem object is removed once the algorithm finishes its run.

4.2 New algorithms

User may implement his/hers algorithm. It is necessary to add this algorithm to Database to be visible in algorithm selection window, add "case" of this algorithm to plan creator, and add static class of this algorithm under the algorithms. If its input is Problem object and output is a Plan object, no further changes are necessary.

5 Testing Tools

Testing tool component enables user to run experiment runs with combinations of settings on precomputed map, outputing results into csv.
The name of the component is BT-Tools and it is accessable via testing mode button in main application window.
Beware, its parametres must be set in code, it does not take any input. Rather, overview of experiment is displayed and upon pressing Start button, the experiments beggin.
The experiments take combinations of different algorihms, number of agents , number of orders and their frequencies.
The algorithm are predefined by user, the agents are randomly generated (based on a seed which is part of the output for better rereplicability), and orders are assigned Start time based on frequencies.
Upon solving one instance, selected information are printed into csv file and the application proceeds to the next instance, until all predefined combinations are solved.
The default output format is following:
MapName, ScenarioName, Algorithm, Agents,Orders,frequency, MakeSpan, RunTime, Movements, Waitings, Cost, Seed.

It can be easily modified to display more/less information about the run via the testing component.