Search and Planning - Project Solution Summary

Author: Diogo Lopes - ist196732

Python Files:

My approach to create this tool was to separate different functions to different files.

proj.py – File with main function. It also has the solution parsing function because my approach revolves around checking if the model is satisfied with a makespan value and if it’s not (the solution file says “unsatisfiable”), the tool runs the model again with the makespan parameter incremented by 1.

parse\_graph.py – Writes the number of vertexes, the vertex with the most adjacent vertexes and the list of edges to a .dzn file. The index of the list identifies the vertex and its value is a list of adjacent vertexes. Outputs to “data/graph.dzn”

parse\_scenario.py – Writes the number of agents, their initial and goal positions and an estimative of the worst-case max number of moves required for them to reach their destination. Outputs to “data/scenario.dzn”

parse\_minspan.py – Writes the minimum number of moves required for all agents to reach the goal. It’s first calculated by running a BFS through all the agents and giving the model the length of longest path, then it’s incremented every time the model fails with this value (this occurs in the parsing\_solution function). Outputs to “data/minspan.dzn”

bfs.py – Simple BFS adapted to receive a dictionary of edges, a starting and goal position. It also as the function to calculate the longest path of all agents.

The tool creates a temp\_solution.txt file because the normal minizinc output, comes out with 2 lines at the end, which are removed, and the rest of the solution goes to the specified file when the program is called.

Model:

Constraints – there’s a single constraint clause with all the limitations. It follows this format:

The first move is equal to the initial position

AND

All agents are currently in different positions.

AND

(An agent doesn’t move in-between steps.

OR

(There’s an edge for the agent to move when it changes position.

AND

An agent doesn’t move to a position that was occupied in the previous step.))

AND

All moves differ from one another.

AND

After the goal is reached, all agents stay in the same vertexes.

Observations:

When making the tools, the biggest bottlenecks in its performance (that I noticed) where that the bigger the difference from the estimate to the real value of the minimum makespan, the worst performance it has because, every time it fails, it must recalculate all the moves from scratch. When there’s only one free vertex to move to (sliding puzzle), the estimate is nowhere near the real value.