SEARCH AND PLANNING PROJECT - 2022/2023

Álvaro Saldanha, 92416
Vasco Cabral, 92568

Summary of Proposed Solution - MAPF w/ CSP

The proposed solution for the specified MAPF with CSP problem attempts to find an optimal solution for the paths, given a graph and a scenario, in which potentially multiple agents in their starting nodes reach their goal nodes, minimizing the number of timesteps needed for them to do so according to a set of constraints and rules.
The implementation, developed with Python and Minizinc, starts by parsing the graph and scenario input files using a new graph object to store their information. Following this initialization, the essential logic behind the implementation rests on continuously increasing the maximum time step by 2 (with the goal of skipping time steps, potentially removing unnecessary processing) and, on each iteration, calling minizinc with the respective graph and scenario and checking to see if a solution has been found. If the current instance of the minizinc model returns as satisfiable, the proposed agent paths are temporarily saved and the maximum time step is decremented by 1. The model is run again in order to check if the previously found solution is indeed an optimal one, or if the decremented and untested time step also presents a solution, therefore being optimal. If this is the case, then the output refers to the last found solution. Otherwise, it's the first one found.
In Minizinc, the developed model has six domain variables, all passed from the Python script:

- The current maximum time step
- The number of nodes in the graph
- The number of agents in the scenario
- The initial agent positions
- The goal agent positions
- The adjacency matrix of the graph

One decision variable, position_at_ts, consists of a matrix of size (current maximum time step, number of agents), saving the position each agent is at a given time step (the node number) and, if complete, their paths from initial position to goal position.
Finally, there are five constraints:

- The positions of agents at timestep 1, in position_at_ts, must be the same as the ones in the respective domain variable.
- The positions of agents at timestep current maximum timestep, in position_at_ts, must be the same as the ones in the respective domain variable.
- At each timestep in position_at_ts, all agent positions have to be different.
- Agent position at timestep ts is empty or the same as in ts-1.
- Agent position at timestep ts is adjacent or same as timestep ts-1.

If the model is satisfiable, then the result will return position_at_ts, which now holds the path for each agent, which complies with the current scenario and all the above restrictions, in respect to the given maximum timestep. This way, by iteratively increasing the timestep by 2 and testing the model, if a solution is found, it's easy to see that either it or the one potentially found by running the model again for timestep - 1, is optimal.

The increment chosen is 2 because it reduces the amount of instances tested to approximately half (compared to testing all time steps). Selecting a higher increment could result in a significant performance decrease, since increasingly complex models would be needlessly tested for cases when the time step 'jump' is too large. The increase in time to compute a solution isn't linear regarding the number of timesteps. So, just an increment of 1 can lead to a steep complexity increase.
An attempted optimization strategy was , on each execution of minizinc, selecting the agent that in a given timestep is more constrained, i.e., that has the smaller domain set, to be searched first. In our testing, this didn't lead to a significant improvement in overall execution time, and was then removed from the code. One last important thing to note is that we have used the chuffed solver, a solver based on lazy clause generation.