Grupo: 05

MAPF with CSP

A estrutura de dados que utilizámos para representar as variáveis do problema no minizinc é uma matriz em que as linhas representam os timestamps e as colunas representam os agentes. Por exemplo a posição do agente j no timestamp i é representado por pos[i,j].

As constantes que utilizámos na resolução do problema foram:

- start - representa as posições iniciais dos agentes;
- goal - representa as posições finais dos agentes;
- n_agents - número de agentes;
- n_vertices – número de vértices;
- makespan – tempo que demora a resolver o problema, ou seja, número de linhas da matriz;
- adj – matriz de adjacências onde cada linha representa um vértice e as colunas são os vértices a que este tem ligação;
- max_adj – o número máximo de ligações em algum vértice do grafo. Os vértices que não têm este número de ligações têm o resto das suas posições preenchidas por 0.
- dist – matriz em que cada linha representa um vértice e as colunas são os tempos mínimos para chegar aos objetivos a partir deste vértice.

As restrições definidas para a resolução deste problema foram:

- pos[1,..] == start e pos[makespan,..] == goal onde o inicio e o fim ficam definidos;
- 3ª restrição para garantir que no mesmo timestamp não existe mais que um agente num mesmo vértice;
- 4ª restrição onde garantimos que caso o agente tenha mudado de posição a ligação que ele utilizou para isso existe usando a matriz adj. E por fim vemos se o agente não foi para um vértice ocupado no timestamp anterior evitando assim o swapping conflict.
- 5ª restrição foi utilizada para otimizar os tempos de resolução uma vez que aqui verificamos que caso o tempo mínimo para um objetivo(dist) for maior do que o makespan – timestamp atual então não vale a continuar a procura nesta solução. Por isso garantimos que esta relação seja menor ou igual.

Em relação ao ficheiro python utilizamos uma bfs implementada por nós para calcular a matriz de distancias (dist) que utilizamos posteriormente para calcular os bounds do makespan. O lowerbound é o maior caminho entre os caminhos mais curtos dos agentes e o upperbound é a soma dos caminhos de todos. Isto acontece quase sempre menos para os casos em que n_vertices – n_agents <= 2 aqui o lowerbound passa a ser a soma dos caminhos de todos os agentes e o upperbound passa a ser o lowerbound * 2. A matriz adj é calculada a partir da lista de edges dada como input.

No final para resolver o problema fazemos um ciclo em que começamos com o makespan = lowerbound e chamamos o modelo do minizinc se o problema não for resolvido incrementamos o makespan. Quando o problema é resolvido saímos do ciclo e damos print do output.