[Info M1 2IS] - Computational Intelligence

Project 2: Reasoning

Six-Nations scheduling

Authors:
Safadi Tasnim
Liu Wenjie
Syed Hasseb Ahmad
Dahmani Kheira

April 15, 2022

1 Introduction

The objective of this project is to face the problem of scheduling matches for a famous double round robin rugby tournament, to tackle this problem, we first formulated our problem in two different approaches Standard Search and CSP. (to be continued).

We will further describe our CSP and search models implemented in order to run our tests. The search models include DFS, BFS, and A*.

2 Constraint Satisfaction Problem (CSP)

2.1 General description

A constraint satisfaction problem or CSP is a problem in which we define the variables and values and bound them by constraints. A solution is then calculated when these variables satisfy all these constraints.

The use of CSPs is attributed to the 1970s where they were used to solve computational problems encountered in image processing. They were then evolved giving rise to the current general framework which is used to satisfy a variety of problems including the famous sudoku and eight queens problem. CSPs are also being used in many different areas such as artificial intelligence and database systems to circuit design, network optimization, and the theory of programming languages.

In our case of 6-nations rugby tournament we defined our CSP using the variables, values and constraints, the details of which are as follows.

2.1.1 Variables

There are 30 variables, with each variable being a match of home vs away team. So France vs Italy will be a different variable from Italy vs France. Further we divided the variables in the form of rounds, so each round would exactly contain three variables.

- **Variable** = {'A': Away Team, 'H': Home Team }
- **Round** = [Variable 1, Variable 2, Variable 3]

2.1.2 Values

The values are the 6 teams participating in the tournament i.e. FRANCE, ITALY, SCOTLAND, IRELAND, WALES, ENGLAND. However, since we have Home vs Away, so we will have 6 values for home and 6 values for away in a match. This makes a total of 12 values as follows:

- **"Home"**, ["FRANCE", "ITALY", "SCOTLAND", "IRELAND", "WALES", "ENGLAND"]
- **"Away"**, ["FRANCE", "ITALY", "SCOTLAND", "IRELAND", "WALES", "ENGLAND"]

2.1.3 Constraints

The rules mentioned in the problem statement were taken as constraints. For a round i match m and team t, these are as follows:

- No two teams can play twice at home in consecutive rounds, or twice away in consecutive rounds , $0 < i < 7 m_i(t_j, t_k)! = m_i + 1(t_j, t_l) and m_i(t_j, t_k)! = m_i + 1(t_l, t_k)$

- The tournament lasts over 10 rounds and uses the following mirroring scheme: $(1, 6)(2, 7)(3, 8)(4, 9)(5, 10), 0 < i < 6, m_i + 5(t_j, t_k) = m_i(t_k, t_j)$
- No team plays against ITALY or FRANCE away twice in consecutive rounds. For all$i$, $0 < i < 7, m_i(it, t_k)! = m_i + 1(fr, t_k) and m_i(fr, t_k)! = m_i + 1(it, t_k)$
- On the last day ENGLAND plays against FRANCE. $i = 5$, $m_i = (en, fr)$ or $m_i = (fr, en)$
- On the first day IRELAND and ITALY cannot play home. $i = 1$ and all $0 < k < 7, m_i! = (it, t_k)$ and $m_i! = (ir, t_k)$

2.1.4 Constraint Graph

All the matches in a round are constrained with each other. Each of these matches is also constrained to all the matches in the next round. The trend continues till the 10th round.

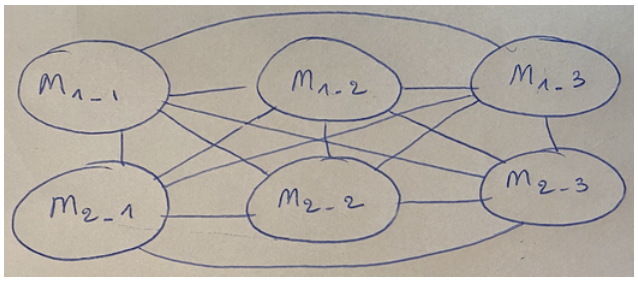

Figure 1: Constraint graph

2.2 Discussion

In our case we implemented all these constraints according to each round. When round 1 was created, all the concerning constraints were implemented until the round 1 is filled with three matches or variables. For each round when a constraint was not satisfied, iterations were made until the best solution for that round was found. In this way 5 rounds were generated and for each round its twin round was generated accordingly.

We were able to successfully implement all the constraints but to make the solution work every time, we had to modify the first constraint. So, we exempted the round 4 and round 5 including their twin rounds from constraint 1 and modified the constraint such that no team plays away in consecutive rounds.

3 Search

3.1 General description

We decided to solve the problem for 5 rounds of the tournament. We simply mirror it at the end. Let matches be the set of all matches (contains 30 elements).

3.1.1 Possible states

A tournament with any number of matches that are compliant with the constraints.

In the code, a tournament is a list containing rounds (lists of maximum size 3). Each round contains matches (lists of exactly size 2). Each match has in the first index the team playing at home and in second index, the team playing against it.

A tournament example would be T = [[['FR', 'IT'], ['SC', 'IR'], ['WA', 'EN']], [['EN', 'SC']]]

- **Initial state:** An empty tournament: []
- **Final state:** T is a valid tournament of size 15 (goal)

3.1.2 Actions

Adding a new match to T from matches

$T' = T \cup m$, where T is a valid tournament, m in matches and T' is still a valid tournament

3.1.3 Branching factor

Since an action is adding a match, the branching factor is the size of matches (in this case 30). However, in reality, we will have less and less branches to explore as we go down in the tree as some matches are added and cannot be added again.

3.1.4 Depth

As the maximum size of a tournament is 15 and we are sure to add a match each time we expand a node, the depth is 15

3.2 Discussion

As shown on the search tree, we start with an empty tournament and stop when the tournament contains 15 matches. Several elements were taken into consideration in order to optimize the search and avoid unnecessary expansions.

- Matches that were already played will not be put in the tournament again and therefore tournaments with duplicate matches are not expanded.
- Similarly, a match's mirror will not be added to the tournament as we will be adding it in the end, when we mirror the tournament.
- Any match addition which directly breaks a constraint will not be an expanded node.

By adding these elements to the code, we were able to significantly increase the performance of our algorithms. Let's analyze the different paths chosen by each search now.

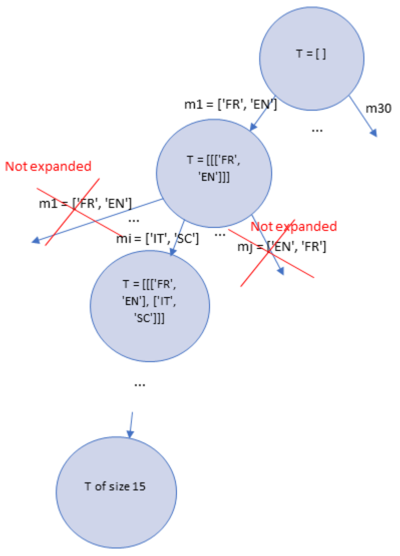

Figure 2: Search tree

3.2.1 BFS

Breadth-first search will expand all the possible nodes on the current depth before moving to the next depth. This was achieved by using a FIFO queue. Given the fact that our branching factor can be as high as 30 this is clearly not the optimal algorithm for our model. At a depth i (where i in [0, 15]), the number of nodes to visit will be 30i. This means that we would have to expand 1.4348907e+22 nodes before we can even obtain a completed tournament (containing 15 matches) and then find one which respects the constraints. For this reason, this algorithm was not run with tournaments of 5 rounds, but rather smaller tournaments.

3.2.2 DFS

Depth-first search will try to search for solutions as deep in depth as it can. This was achieved by using a LIFO stack. It will stop as soon as it has a completed valid tournament. As there are probably multiple valid tournament organizations possible, it is much less likely that it will need to go through all the branches like BFS. In addition, given the fact that no tournament organization is "better" than another (as long as all constraints are satisfied it is valid), DFS will actually provide us with an optimal solution.

3.2.3 A*

A* will associate each node with a priority value. Nodes with the lowest value will be prioritized and explored first. This is done using a priority queue. The priority value is f(x) where:

- f(x) is the sum of g(x) and h(x).
- g(x) is the cost of an action to move from one node to another (which is a branch).
- h(x) is the estimated cost to move from one node to the goal state.

In our model, there is no cost function. Since we only consider valid tournaments, adding any match is as good as adding another, therefore if we wanted to use a cost function, it would be constant. This is why we decided to leave it as equal to 0. Unlike scenarios where, for instance, we would be trying to minimize a score or total steps, we are not confronted with this (we will always have to put 15 matches to find a solution).

Heuristic:
In order to find a descent heuristic, we tried answering two questions:

- "How far am I from reaching my goal given any uncompleted tournament?"
- "How can I know from a selection of uncompleted tournaments, which one is more likely to be completed and reach a goal?"

For the first part, we know that the goal is a tournament of size 15, so for any given tournament, we can compute the minimum remaining steps as: $h_1(T) = size(goal) - size(T)$, where T is any given tournament and $goal$ is the state goal. Since we are trying to minimize this function as much as possible, we eventually reach $h_1(T) = 0$. This heuristic is admissible since it is always underestimating or equal to its actual "distance" from the goal tournament.

Later on, we thought about trying to predict upcoming constraints given a tournament state. We could only find prediction regarding the last round's constraint: England must play against France. In fact, although put England vs France match in a round that isn't the last is not a direct violation, it was inevitably leading to a tournament with no solution so defined. Therefore we have $h_2(T) = 100$ if the match England vs France is played before round 5, $h_2(T) = 0$ otherwise.

There were no other heuristics we could think of in order further promote exploring certain tournament instead of others. However, we decided to use heuristic to "optimize" the solution. This is in contradiction with what was said before; there is no solution that is better than another solution. This is true given the statement of the project, we can either meet the constraints or we can't. However, in the real world, perhaps we would still want to keep some constraints even if they will not be respected in each round. Since the most restrictive constraint was "No team plays against ITALY or FRANCE away twice in consecutive rounds", we decided to find solutions which minimize the number of times this constraint is not respected. Therefore we have $h_3(T) = \frac{constraint(T)}{3}$, where constraint(T) returns the number of times the previously stated constraint is violated. The division was mandatory in order to prevent the algorithm from testing every possible match combination as it would give too much importance to this heuristic[1]. We picked 3 as any lower number gave the heuristic too much importance (leading to unreasonable waiting time) and any higher number didn't not provide us with any significant change.

3.2.4 Implementation

Five python files are used for the search part:

- actions.py which contains valid action verifications.
- util.py which contains LIFO Stack, FIFO Queue and Priority Queue implementation.
- bfs.py, dfs.py, and astar.py for respectively bfs, dfs, and a*.

4 Testing

4.1 Scenario created

We tried to find a solution that meets all the constraints for 6 teams. Since there is no solution, we gradually liberalize the restrictions in two dimensions: combinations of constrains and number of teams.

- Constraints 1: No two teams can play twice at home in consecutive rounds, or twice away in consecutive rounds
- Constraints 2: No team plays against ITALY or FRANCE away twice in consecutive rounds
- Constraints 3: On the last day ENGLAND plays against France
- Constraints 4: On the first day IRELAND and ITALY cannot play home

	Constraint 1, 2	Constraint 1, 2	Constraint 1, 2	Constraint 1, 2
6 Teams	Scenario 1	Scenario 2	Scenario 3	Scenario 4
5 Teams	Scenario 1	Scenario 1	Scenario 1	Scenario 1
4 Teams	Scenario 1	Scenario 1	Scenario 1	Scenario 1

[1]We could have also multiplied the other heuristics by 3 instead

4.2 Results

Execution time				
	BFS	DFS	A*	CSP
Scenario 1				
Scenario 2				
Scenario 3				

#Nodes explored				
	BFS	DFS	A*	CSP
Scenario 1				
Scenario 2				
Scenario 3				

*If two teams cannot play twice away in consecutive rounds, they cannot play against ITALY or FRANCE away twice in consecutive rounds. So, constraints 1 includes constraints 2.

4.3 Discussion

5 Conclusion

In regards of the results issued from the previous scenarios, –¿ conclude which constraints should be removed and which set of teams works best towards a final possible solution (to be continued)