CAB203 Graphs Project

Braydan Newman: n11272031

May 21, 2023

1 Tournament structure

The Tournament Structure (TS) is defined as S, a set of games, and P, a set of players. Where each game is the combination of 2 different players (a, b) representing a game between a and b. The graph is symmetric such that a game (a, b) is the same as the game (b, a).

$$S = \{s_1, s_2, \cdots, s_n\} \tag{1}$$

$$P = \{p_1, p_2, \cdots, p_n\} \tag{2}$$

The objective is to test whether the given TS is valid which is determined by the following the following property:

For every pair of distinct players A, B, either A plays against B or A plays against some other player that plays against B.

The graph for TS can be modelled as a undirected graph $G = (V, G)$, such that the vertices V is the set of all distinct players and the edges E is a set of games represented as a pairs of players. For any two distinct vertices a and b ($a \neq b$) in V, either there exists a direct edge in E between A and B:

$$\forall a, b \in V \tag{3}$$

$$(a, b) \in E \tag{4}$$

Or there exists a vertices c that has both a and b as neighbours.

$$\exists c \in V : (a, c) \in E \wedge (b, c) \in E \tag{5}$$

The solution to this problem is finding the set N such that each players neighbours of each players neighbours matches V. If this is the case, it is a valid tournament as all players ($players = V$) play a all other players with a maximum of path distance of 2.

The python function `gameOK` implements this solution using the `N` from `graphs.py`. This is used to create a set of neighbours for every vertex in G and all their neighbours. If this set has all the same elements as V, then this is valid returning `true` otherwise this returns `false`.

2 Potential referees

Now the referees of each game need to be decided. Each referee can not have any conflicts in the game, such as the referee can not be apart of the game nor can they have a conflict with the players. The potential referees and conflicts for the referees are given in a CSV file, while the game is given as player 1 and player 2 $S = \{s_1, s_2\}$. Using set theory we can model the potential referees conflicts as a set P so that an element of P is (r, C), such that r is the referee and C is set of conflicts. U will be the set for valid referees of the game S.

$$S = \{s_1, s_2\} \tag{6}$$

$$P = \{p_1, p_2, \cdots, p_n\} \tag{7}$$

$$C = \{c_1, c_2, \cdots, c_n\} \tag{8}$$

$$U = \{u_1, u_2, \cdots, u_n\} \tag{9}$$

The objective is to create the set U with all the referees P that don't have a conflict C with the game S.

To determine if a potential referee can referee a game they can't be apart of the game:

$$U1 = \{\forall (r, C) \in P : r \notin S\} \tag{10}$$

The referee also must have no conflicts of interest with any of the players in the game:

$$U2 = \{\forall (r, C) \in P : C \cap S = \emptyset\} \tag{11}$$

If the referee meets both of these criteria they are a valid potential referee for the game:

$$U = U1 \cap U2 \tag{12}$$

As all elements in U have to be in both $U1$ and $U2$, to have no conflicts.

The python function `potentialReferees` takes 3 variables, `csv_filename`, `player_1`and `player_2`. After opening the csv file and reading all the contents into memory, a simple set comprehension which checks all referees and there conflicts against the players, returning a set of valid potential referees.

3 Assign referees

From the Potential Referees we must now assign one referee to every game and at most 1 game to a referee. The games and referees come as a bi-partite graph defined as $G = (V, E)$ with S as the games and R as the referees, making up the two sides on bi-partite graph and E is the edges that define that a referee r is a potential match for a game s, $e = (s, r)$.

$$S = \{p_1, p_2, \cdots, p_n\} \tag{13}$$

$$R = \{r_1, r_2, \cdots, r_n\} \tag{14}$$

$$E = \{e_1, e_2, \cdots, e_n\} \tag{15}$$

$$V = \{S \cup R\} \tag{16}$$

$$G = (V, E) \tag{17}$$

From this we can find the maximum matching of the graph G as M. M Defines another bi-partite graph such that each each game s is disjoint from all other games. M is a subset of G:

$$M \subseteq S \tag{18}$$

If M is the same length as S then we know that every game has been assigned a referee but if the length is not equal then a game has been missed and a game to referee can not be found.

$$|M| = |S| \tag{19}$$

The python function `gameReferees` takes in the graph `game_potential_referees` and formatting the bi-partite graph into 2 sets representing games and referees and creating the set for edges between. We use the given function `maxMatching` from `digraphs` to find the max matching bi-graph. From this we determine if all games are accounted for, if so we return the graph if not we return `None`.

4 Games schedule

Now the games all have referees to games, we have to schedule them but some games contain the same people. We can't have the same person at two different games at the same time, but must also do this in the minimum amount of time slots. The vertices V can be a set such that each element is all the people in a game (referee and players) $v = (player_1, player_2, referee)$. While edges E is connection between vertices that have a shared people. This creates a graph $G = (V, E)$.

$$G = (V, E) \tag{20}$$

$$V = \{v_1, v_2, \cdots, v_n\} \tag{21}$$

$$E = \{e_1, e_2, \cdots, e_n\} \tag{22}$$

From here we colour the graph with a minimum colouring algorithm so that every vertex has a different colour from all its neighbours, and there is the lowest number of different colours possible. This allows us to sort the games into different colour groups, each group having no people more then once. Each one of these groups is a time slot and each game in the time slot can all be played at the same time.

The python function `gameSchedule` takes the graph as a bi-graph of referees to games and turns it into the a set of people in each game `V` and a set with an edges where if two games have a person in common `E`. We use the given function `minColouring` from `graphs` and input `V and E`. With the output of that function we group all games with common colours as each time slot and return a list of time slots with a set of games in each time slot.

5 Player ranking

After each game is finished there is a winner and a loser, a set of all players V and with a set of these outcomes E each element representing the winner and loser $e = (w, l)$. We use the game outcome for the edges of the graph G so that we get edges going from winner to loser. With the graph being a directed graph we can use Topological Ordering to put each game in order and because each edge was winner to loser the order is from highest ranking to lowest ranking.

$$G = (V, E) \tag{23}$$

$$V = \{v_1, v_2, \cdots, v_n\} \tag{24}$$

$$E = \{e_1, e_2, \cdots, e_n\} \tag{25}$$

$$e = (winner, loser) \tag{26}$$

The python function `ranking` takes in a set of game outcome tuples as (winner, loser). From this set a new set is created with all the unique players in it. We use the given function `topOrdering` from `digraphs` with the graph from both of these sets the output is a list of players in order of ranking.

References

[1] *CAB203 Tutorial Solutions 9.4.* `https://canvas.qut.edu.au/courses/1979/pages/9-dot-4-tutorial-solutions?module_item_id=1192666` QUT, 2023.

[2] *CAB203 Tutorial Solutions 8.4.* `https://canvas.qut.edu.au/courses/1979/pages/8-dot-4-tutorial-solutions?module_item_id=1192637` QUT, 2023.

[3] *CAB203 Tutorial Solutions 8.4.* `https://canvas.qut.edu.au/courses/1979/pages/7-dot-4-tutorial-solutions?module_item_id=1189205` QUT, 2023.