

# CAB203 Graphs Project

Joshua Włodarczyk: N11275561

Date: May 27<sup>th</sup>, 2024

## Tournament Structure:

The first problems ask to develop a structure for the tournament; with the intended purposing being to create a formula that can determine whether a set of games meet the required conditions set of every player having an appropriate opponent.

The tournament structure can be broken down into 2 main properties:

1. Every player plays against each other or there are a minimum of 2 other players they both play against.
2. All players have the same number of games.

The first property can be defined as for every player ( $p$ ) in the set of games ( $S$ ) is the neighbour of all other  $p$ , which is defined as possible opponents ( $o$ ) or ( $U$ )  $p$  shares at least 2 neighbours with the  $o$  if they are not neighbours.

The second property can be defined as for every  $S$  find the number of neighbours for ( $p, o$ ) and ensure they each have an equal amount.

Both properties involve the formula of finding the neighbours of ( $p, o$ ).

$$N_G(u) = \{v \in V: (u, v) \in E\}$$

The degrees formula can be used to determine the number of neighbours per element in  $S$  and Handshaking lemma can be used to help determine ( $p, o$ ) all have at least 2 neighbours in common.

$$d(u) = |N_u|$$

$$2|E| = \sum_{u \in V} d(u)$$

From the stated properties I can be determined that the tournament structure needs to be symmetric and irreflexive in nature. (Reference 2)

The following graph depicts a game that meets the tournament structures requirements:

Figure 1:

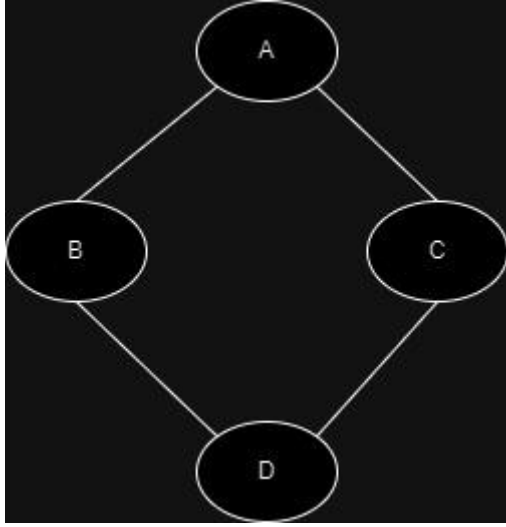


Figure 1 depicts the following:

$A = a, B = b, C = c, D = d$

$$a = p$$

$$N_G(a) \cup \sum_{a \in V} d(a) = o$$

The vertices are the players, and the edges are the opponents (neighbours) the players can play with, thus forming a valid game.

Neighbours:	Degrees:
$N_A = \{ B, C \}$	$d(A) = 2$
$N_B = \{ A, D \}$	$d(B) = 2$
$N_C = \{ A, D \}$	$d(C) = 2$
$N_D = \{ B, C \}$	$d(D) = 2$

With this the properties can be given as the following 3 equation formula:

Let  $V$  be the set of vertices, let  $E$  be players to their opponent (neighbours) and let  $S$  be the set of all players. Let  $p$  be the player and let  $o$  be the opponent (neighbour).

Equation 1, from the set of vertices make sure each player has an opponent.

$$S_G = \{(p, o): p \in V\}$$

Equation 2, all vertices have at least 2 players to play against or are all opponents of each other.

$$V = \bigcup_{(p,o) \in E} \{N_G(p): o \in S\}$$

Equation 3, all edges are determined by whether the requirements of equation 2 are met and each player plays at most one game.

$$E = \{p \in S: \sum_{p \in V} d(p)\}$$

When implemented into python with  $V$  determining the neighbours of the players with  $E$  counting the number of games and determining checking if the  $p$  play against all  $o$  or share  $o$  in common with other  $p$  that are not  $o$ .  $V$  and  $E$  are sets with  $S_G$  representing a Boolean statement that checks if the conditions of  $V \cup E$  are met. The edges are made undirected following the code from tutorial 7 (Reference 6). The function  $N$  and  $degree$  from the *graphs.py* module (Reference 4) is used with the  $V$  and  $E$  to determine the listed properties. If true, then all properties are met, and the game is valid otherwise return false.

## Potential referees:

The second problem requires that each referee is assigned a game to referee with some of the referees being players themselves. A formula meeting the listed 4 properties below will need to produce a set of games with assigned referees:

1. Every game needs at most one referee.
2. Every referee needs at most one game.
3. A Referee can't be a player in a game they are playing in.
4. Referee can't have a conflict in that game.

Graph bipartite theory is present in developing a formula as the players need to be separated from the referees in each game games (Reference 2). Graph theory follows the following formula:

$$G = (V, E)$$

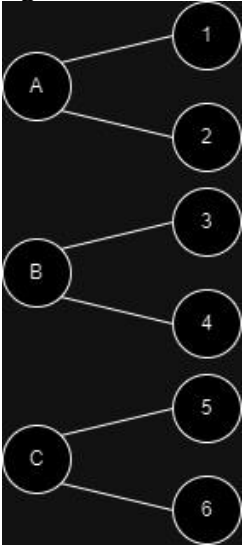
The bipartition of 2 elements represented by A and B:

$$A = \bigcup_{j \text{ odd}} D_j, \quad B = \bigcup_{j \text{ even}} D_j$$

Matching of the bipartite graph will also be used to produce a set of matched games:

$$G = (V, E) \text{ is a subset } M \subseteq E$$

The following figure depicts the structure of the games that needs to be produce:

<p><b>Figure 2:</b></p> 	<p>Figure 2 depicts a bipartite graph depicts the ideal set of games with the letters representing the referees and the numbers representing the players.</p> $N_A = \{1, 2\}$ $N_B = \{3, 4\}$ $N_C = \{5, 6\}$ <p>The neighbours of the referees are the players and show how the games should be formatted.</p>	<p>The vertices, <math>V</math>, are the set of all players and referees.</p> <p>The edges, <math>E</math>, are the set of games, the first player and second player in that game, to referees.</p> <p>The formula can be broken down into 3 equations:</p> <p><math>S</math> is the set of all games.</p> <p><math>R</math> is the set of all referees.</p> <p><math>C_r</math> is the set of conflict for each referee.</p> <p><math>t_{ij}</math> represents whether <math>R_j</math> is assigned to <math>R_i</math></p> <p>Equation 1: All games need at most one referee and each referee needs at most one game.</p> $\sum_{i=1}^{ S } \sum_{j=1}^{ R } t_{ij} \leq 1: \forall r_{ij} \in E$ <p>Equation 2: For each referee in the set of the games, do they have any conflicts and are they present in that game.</p> $\forall S: R_x \notin C_r$ <p>Equation 3: All referees who are a player (neighbour) in a game must not be able to ref that game.</p> $\exists t_{ij}: R_x \notin S_x$
---	--	---

The python code follows the structure of the formulas with the CSV file containing the referee conflicts being read to create a dictionary of referees (key) and player conflicts (value), this code follows the example in lecture 6 Python CSV DirectReader section (Reference 1).  $V$  (vertices) is a set of all games and referees and  $E$  (edges) being games to referees for games that don't have player conflicts with the referee in them.  $V$  and  $E$  is used with the *bipartition* function from *graphs.py* (Reference 4) the output of this is used for the function *maxMatching* from *digraphs.py* (Reference 5) with  $E$  the result is then changed into a dictionary which is returned, the referee is the value and the game as the key.

## Assign referees:

Problem three states that a formula needs to be developed that can schedule games into a tuple such that the first element is game 1 and the second element is game 2.

The assigned referee games that problem 2's formula produces will set the games that need to be grouped. The formula will need to meet the 3 following properties:

1. Each person is involved in at most one game in any game group as player or referee.
2. Each game must have a different referee.
3. Each game is only grouped at most once.

Graph colour theory is present in developing a formula as the *chromatic* number can be used as a representation for the minimum number of possible timeslots that can be generated from assigned referees (Reference 2). Graph theory follows the following formula:

$$G = (V, E)$$

With the following formula representing that no 2 vertices are adjacent:

$$U \subseteq V$$

The following formula can be used for the *partition* of the Set:

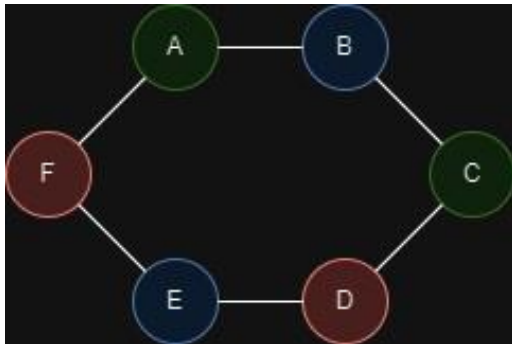
$$S = \cup_{j=1}^n S_j$$

$$S_j \cap S_k = \emptyset$$

$$\forall j, k \ j \neq k$$

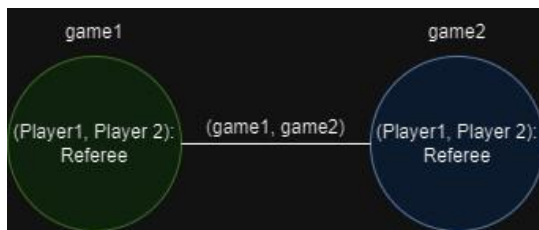
The Following graph is a depiction of the structure of what the problem follows:

Figure 3:



Note that vertices with the same colours have different players and referees.

Figure 4:



Note figure 4 displays that the edge has been formed as either player1 or player 2 in game1 is the same in game2 or they have the same referee.

Figure 3 is a *coloured undirected* graph displaying how the games should be organised into timeslots. A, B, C, D, E, F each represent a different game (set of players) and are the vertices. The vertices are the keys of assigned referees (output of problem 2), each key is an individual game.

$$V = (player1, player 2)$$

The edges represent the games that share the same referees or players. It indicates what can't games can't be scheduled at the same time.

$$E = (game1, game2)$$

As previously mentioned, the colours display the *chromatic* number and determine the number of timeslots, in this case there are 3.

*Green: Timeslot 0*

*Blue: Timeslot 1*

*Red: Timeslot 2*

The Formula can be stated as:

Let  $P$  be the set of players and referees that are playing.

Let  $S$  be the set of all games.

Let  $R(g)$  be the referees assigned to each game.

Let  $P(g)$  be the referees assigned to each game.

Let  $S_x$  be the game groups where  $x$  is the game in  $G$ .

Each person is in at most one game group as player or referee:

$$\forall S_x \subseteq S, \forall p \in P: \{g \in S_x \mid |P(g) \cup R(g)| \leq 1\}$$

Each game group is group once and has a different referee.

$$\{|S_x|g \in S_x \leq 1\} \bigcap_{\forall g_x \in G} \{P(g_1) \neq P(g_2), R(g_1) \neq R(g_2)\}$$

The python code closely follows the explanation and graph structure with elements such as  $V$  and  $E$  being sets and games and refs being used as list to check that each referee and game meets the set requirements of Equation 1 and 2.  $V$  and  $E$  are used with *minColouring* from the *graphs.py* module (Reference 4). The  $C$  represented as the chromatic colouring of the graph is used to produce a list of dictionaries with *colourClassesFromColouring* from the same python module, all the elements of this list are extracted from this and returned.

## Games schedule:

The fourth problem wants to order the game schedule produce from the previous problem so that referees who are players play in the first game group. The problem can be broken down into one simple property:

1. Players who are playing and referring other games play first.

From this it can be determined that the problem is a directed graph with the start point being the referee that will ref the first game group that need to be played first and the end point being the last game group played. The start point game group will have players that are also are referees, with the end point will have players that have no games to referee.

Directed graph theory will be used as the problem is a directed graph:

$$G = (V, E)$$

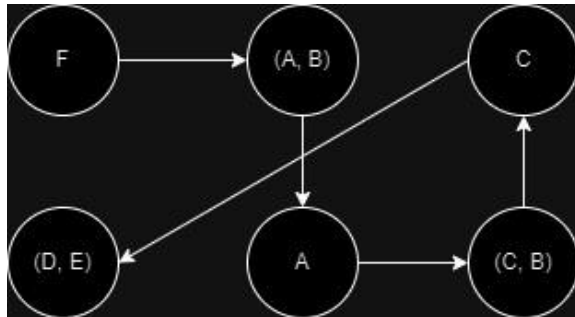
$$\forall v \in V, (v, v) \notin E$$

With the directed graph topological ordering can be used to state the order of the vertices in the directed graph (Reference 3):

$$R = \bigcup_{j=1}^{|S|} \{(s_j, s_k) : k \in \{j, \dots, |S|\}\}$$

The following figure displays a basic graphical structure of the problem:

Figure 5:



*assignedReferees*

$$= \{(A, B): F, (C, B): A, (D, E), C\}$$

*F, A, C*

*= Referees from assignedReferees*

$$\{(A, B)\}, \{(C, B)\}, \{(D, E)\} = \text{gameGroups}$$

Note: the example uses game groups with a game in each. The graph structure would still work if the game groups had 2 games.

*V* is represented as all the referees and game groups.

*E* is the direction from the referee to the game group to the referee that plays in that game group.

Figure 5 depicts an *asymmetric directed* graph of the order game groups should be played based on the game referees play.

The constructed set from the figure:

$$G_0 = \emptyset$$

$$V_0 = \{A, C, F, (A, B), (C, B), (D, E)\}$$

$$G_1 = \{F\}$$

$$V_1 = \{A, C, (A, B), (C, B), (D, E)\}$$

...

$$G_4 = \{C\}$$

$$V_4 = \{(D, E)\}$$

$$G_5 = \{(D, E)\}$$

From this we can indicated the graph is *DAG* with the topological ordering being:

$$F, (A, B), A, (C, B), C, (D, E)$$

The topological order displays the order that referees ref a game group, as in *F* refs *A* and *B* then *A* can ref *C* and *B* then so on.



The formula can be written out as:

Let  $R$  be the topological order for the game schedule and let  $S$  be the set of game groups.

Let  $A$  be the set of assigned referee games and let  $T$  be the set of game schedules.

Let  $A(r)$  be the referee assigned to each assigned referee game.

Let  $S_x$  be the game groups where  $x$  is the game in  $G$  and let  $T_x$  be the game groups in game schedule.

From the topological order create a schedule of game groups:

$$\forall (S_1, S_2) \in R: \sum_{i=T_1}^{T_n} S$$

Out of the game schedule,  $T$ , order the game groups such that the game groups that referees play in ref games first and referees with no games ref their games after. The property of the problem can be met, referees who are players play first:

$$R = \bigcup_{j=1}^{|T|} \{((S_1, S_2)): ((S_1, S_2), A(r)) \in \{j, |S|\}\}$$

Implementation of the method into the python code follows by letting variables  $V$  and  $E$  as sets representing vertices and edges. The code checks if the referee is playing or refereeing in each game for all the game groups. Vertices are a set of all game groups and referees. Edges are added from game to referee for the  $E$  set if the referee is in the game group. Edges are added from referee to game group if the referee is a referee in that game. The sets  $V$  and  $E$  are used in the *topOrdering* function from *digraphs.py* (Reference 5) to create a list of all game groups ordered by the previously stated property, this list is then returned.

## Player ranking:

The fifth and final problem looks at finding the maximum possible score for each player with the score coming from the games one by each player. The following problem can be broken down into 4 main properties:

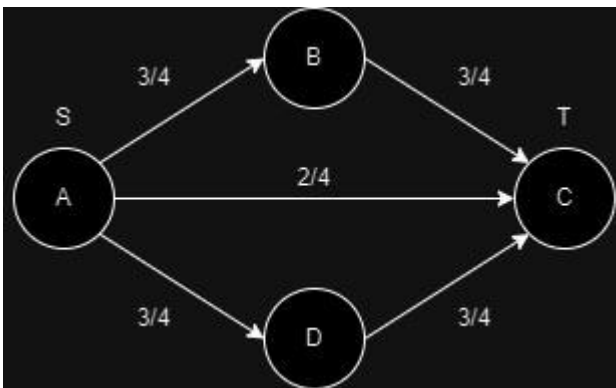
1. If a player wins against their opponent, it is a primary win ( $p$ ), player gains primary number of tokens.
2. If that opponent wins against another opponent the player has not played against and they win it is considered a secondary win ( $s$ ) for the player, opponents gain primary number of tokens and player gains secondary number of tokens.
3. The player does not gain a secondary win from that opponent when they play against other opponents they have played against.
4. A capacity ( $c$ ) is set to determine the maximum number of tokens a player can receive from their opponents.

The following properties can be solved using flow theory. The use of the maximum flow formula into the drain can be used to find the flow output of each game (Reference 3). The formula is as such:

$$\sum_{(u,d) \in E} f((u,d))$$

The following graph depicts the structure of the problem using flow graph theory:

Figure 6:



Vertices:

$A = \text{Alice (S: SOURCE)}$   
 $B = \text{Bob}$   
 $C = \text{Charlie}$   
 $D = \text{Dave (T: SINK)}$

Added flow to get the maximum number of tokens from each player:

$A: 3 + 3 + 2 = 8$   
 $B: 3$   
 $D: 3$   
 $C: 0$

Figure 6 displays a representation of how the scoring should be played out and follows the example of test 1 in TestScores from test\_project.py.

The vertices ( $V$ ) are the set of all players in the game with edges ( $E$ ) being the flow direction of each player from the source to the sink. The flow ( $F$ ) is set by the number of primary wins and secondary points each player gets; this is set to all the corresponding edges that include said player. The capacity ( $c$ ) of each edge in this case 4 states the standard for the maximum number of tokens.

The max scoring for each player is calculated by adding together the flow output for each of the vertices like so:  
 $\frac{3}{4}$  for a primary win  
 $\frac{2}{4}$  for A comes from the 2 points given from the secondary wins granted from B and D victories ( $\frac{1}{4}$  each)

The formula of the problem follow like so:

Let  $V$  be all players in the game and let  $E$  be the edges for  $(u, v)$  to the sink,  $u$  representing the winning player and  $v$  representing the losing opponent in  $E$ .

Let  $P$  be the set of primary wins per player and let  $S$  be the set of secondary wins per player.

Let  $f(u, v)$  represent the flow from player  $u$  to opponent  $v$ .

Primary wins for each player:

$$P = f(u, v)$$

Secondary wins for each player:

$$\forall S: (u, v) \in E \bigcap (v, w) \in E \bigcap (u, w) \notin E$$

Capacity constraint for each player:

$$\sum_{(v,u) \in E} : f(v, u) \leq C$$

Maximum Token amount for each player:

$$T = \sum_{(v,u) \in E} \exists P, S$$

In the python code the maxflow would be used to find the maximum flow amount of each set vertices.  $V$  (vertices) would be the set of players and  $E$  (edges) would be the flow direction of players towards the drain from the sink. The function *augmentedFlow* and *maxFlow* from *digraphs.py* (Reference 5) would be used to augment the flow graph with  $F$ , calculated tokens based off  $p$  and  $s$  for each of edges in  $E$ . *maxFlow* is then used to find the maximum flow of each of the vertices, a dictionary of the maximum flow for each of the vertices is then returned.

## References:

1. Mathew McKague, CAB203 Lecture 6.  
<https://canvas.qut.edu.au/courses/16665/files/3497108/preview>
2. Mathew McKague, CAB203 Lecture 7.  
<https://canvas.qut.edu.au/courses/16665/files/3497116/preview>
3. Mathew McKague, CAB203 Lecture 8.  
<https://canvas.qut.edu.au/courses/16665/files/3497113/preview>
4. Mathew McKague, *graphs.py*.  
<https://canvas.qut.edu.au/courses/16665/files/3812041?wrap=1>
5. Mathew McKague, *digraphs.py*.  
<https://canvas.qut.edu.au/courses/16665/files/3812042?wrap=1>
6. Alan Yu, *tutorial7S2.py*.  
<https://canvas.qut.edu.au/courses/16665/files/3940019?wrap=1>