Algorithmic Game Theory and Applications
Exercise 2

s0840449

1 Question 1

1.1 Part (a)

In the following question, let the nodes be labelled as seen in Figure 1 (i.e. where nodes 1, 4, and 6 belong to Player 1, and nodes 2, 3, and 5 belong to Player 2.) Whenever a payoff is considered, it will be given in the form (u_1, u_2), where u_i is the payoff for Player i.

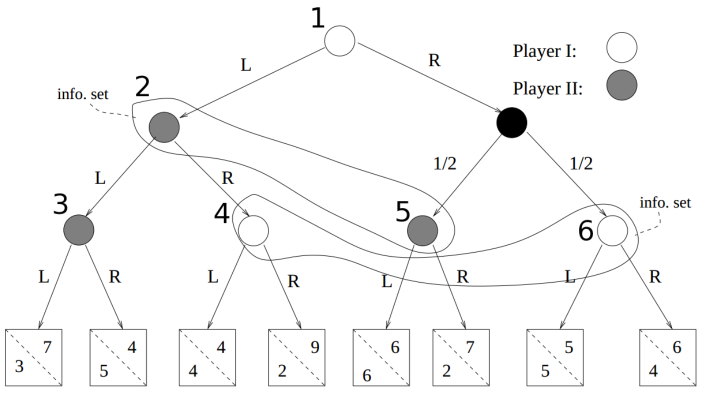

Figure 1: The labelled game tree.

A pure Nash Equilibrium for a finite perfect information extensive form game can be calculated in a bottom up manner, by considering the possible choices at each node starting from the leaves. As there is only one choice at each leaf (the leaf itself), I will start from the first non-leaf level - nodes 3 to 6.

Starting with node 3, the possible actions are $\{L, R\}$, with payoffs of $(3, 7)$ and $(5, 4)$ respectively. As node 3 belongs to Player 2, they will want to maximize their local payoff by choosing L, for a payoff of $(3, 7)$.

Next, consider node 4. Since this node is part of an information set with node 6, Player 1 is unable to tell them apart. Therefore, there are two possible payoffs for each of the action choices $\{L, R\}$, and Player 1 must assume that the minimum payoff for each action is the true payoff - i.e., they must assume the worst case. Choosing L will either result in a payoff of $(4, 4)$ or $(5, 5)$, so Player 1 must assume that the payoff is $(4, 4)$. Choosing R will either result in a payoff of $(2, 9)$ or $(4, 6)$, so Player 1 must assume that the payoff is $(2, 9)$. Therefore, Player 1 will choose L, for a local payoff of $(4, 4)$.

Node 5 is also part of an information set, with node 2. The possible payoffs for the action L are the result of node 3, $(3, 7)$, and $(6, 6)$, so Player 2 must assume that the true payoff is $(6, 6)$. For the R case the possible payoffs are the result of node 4, $(4, 4)$, and $(2, 7)$. Therefore Player 2 must assume that the payoff is $(4, 4)$. Therefore L gives a better payoff, of $(6, 6)$.

As noted, node 6 is part of an information set with node 4, so the action for Player 1 has already been calculated - L, for a payoff of $(4, 4)$.

Moving up a level, node 2 is also part of a previously computed information set with node 5. Therefore, Player 2 will play L, for a payoff of $(6, 6)$.

Finally, node 1 belongs to Player 1 and has two actions, $\{L, R\}$. The payoff for L is the result of node 2, $(6, 6)$. The payoff for R is more complicated, as it is the output of a chance node which selects between nodes 5 and 6 with equal probability. Therefore, the payoff is equal to the sum of the payoffs from nodes 5 and 6, each weighted by $\frac{1}{2}$. This gives a payoff of $(5, 5)$. As $(6, 6)$ is better than this, Player 1 will choose L at node 1, for a payoff of $(6, 6)$.

Putting these together, the strategy functions can be represented as vectors $s_i = (a_1, ..., a_6)$, where a_j is the action taken by Player i at node j, or $\emptyset$ if Player i does not own node j. Then the Nash Equilibrium is:

$$s_1 = (L, \emptyset, \emptyset, L, \emptyset, L)$$
$$s_2 = (\emptyset, L, L, \emptyset, L, \emptyset)$$

Following the relevant player action at each node from the root, the Nash Equilibrium strategy gives the tree LLL, for an expected payoff of $(3, 7)$.

1.2 Part (b)

I.

For this description each node will be labelled by the actions taken to get there. I have assumed that the ability of Monty and the Contestant to randomize is represented as them being allowed to perform mixed strategies, rather than them having access to chance nodes.

The root node, N_0, belongs to Monty, and has Act(N_0) = {A, B, C}. Nodes A, B, and C all belong to the Contestant, and are all part of the same information set, since the Contestant does not know what door Monty chose. Each of these nodes has Act($\{A, B, C\}$) = {A, B, C}. The child nodes depend on how Monty and the Contestants choices have matched up. A node ZX, where Z was Montys choice at N_0 and X was the Contestants choice at node Z, can be in two cases:

- $X = Z$. In this case, Monty can open either of the remaining doors, so ZX has Act(ZX) = {A, B, C} - {X}, that is, it has two children.
- $X \neq Z$. In this case, Monty has only one door he can open, so ZX has Act(ZX) = {A, B, C} - {Z, X}, that is, it has one child.

Let Montys choice of door to open be Y, giving a node ZXY. There are 12 such nodes: AAB, AAC, ABC, ACB, BAC, etc. Importantly, since the Contestant does not know which door the car is behind, these nodes form 6 information sets determined by the last two letters: {AAB, CAB}, {AAC, BAC}, etc. Finally, from each ZXY the contestant has two choices based on which door Y was just opened: {A, B, C} - {Y}. This gives 24 leafs ZXYW. For each leaf, the payoff to Monty (and inversely to the Contestant) depends on if W = Z - if so, the Contestant wins and Monty recieves a payoff of -50, otherwise the Contestant loses and Monty recieves a payoff of 50.

A graphical representation of the game tree is given in FIGURE.

II., III.

To determine the

2 Question 2

2.1 Part (a)

The strategy denoted by bold arrows in Figure 2 is a pure Nash Equilibrium which is not subgame-perfect. This is caused by the fact that Player 2 is making a 'non-credible threat' - deliberately stopping Player 1 from switching by making a sub-optimal move. Player 1 will not unilaterally switch their strategy because the resulting payoff, 0, is less than the current payoff, 3. Player 2 will not unilaterally switch either as the resulting payoff is the same in either case, 4 (since the path where Player 2 has an option is never taken).

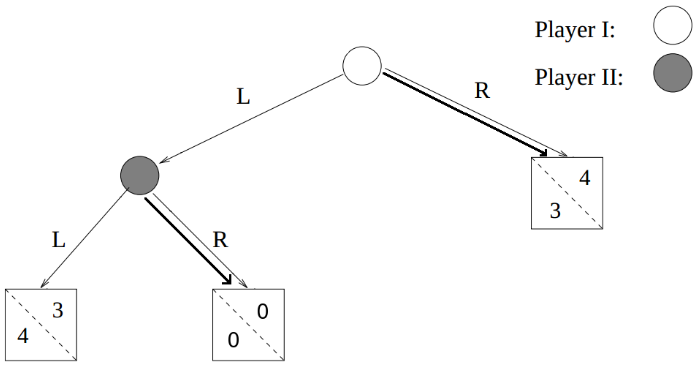

Figure 2: A pure Nash Equilibrium which is not subgame perfect.

2.2 Part (b)

This can be seen by following Kuhn's proof[1] for the existence of a pure Nash Equilibrium (NE) in every finite n-person extensive form (EF) perfect information (PI) game. As every subgame of a finite EF PI game is itself a finite EF PI game, Kuhn's proof also shows the existence of a subgame perfect pure NE. Modifying this proof to include the assumption of no chance nodes and unique terminal payoffs for each player shows that there is only one subgame perfect pure NE. The new hypothesis is that a game G_w, as described in the question, has only one subgame perfect pure NE.

Base case. Let the base case be the game of depth 1 (unlike in Kuhn's proof). As there are no chance nodes in the game, the root w belongs to a player $i > 0$. Each child of w is a leaf node, and each has a distinct payoff for player i (by definition of the game). Therefore, there exists only one pure NE; let player i choose the leaf node wl with the highest payoff for themself, and let all other players strategies at w be empty (as they cannot make a move). This unique NE is subgame perfect as there is only one subgame, the game itself.

Inductive step. Suppose the depth of G_w is k + 1. Each subtree of the game has depth k, and so the induction hypothesis holds - they each have a single subgame perfect NE. Additionally, as the value of a pure Nash Equilibrium in a game with no chance nodes is the value of the leaf following it ends up at, each subtree has a set of payoffs which are distinct for each player (by the definition of the game.)

The next step is to define a strategy $s^w = (s_1^w, ..., s_n^w)$ for the root w. As there are no chance node, w belongs to some player $i > 0$. As noted, each subgame $T_w a_j'$ has a distinct payoff for any one of the players, so has a distinct payoff for player i. Therefore, let i's strategy s_i^w be the strategies in each subtree, $\cup_{a \in Act(w)} s_i^w a$, plus the pure choice $\{w \to a'\}$ where $a' \in Act(w)$ leads to the (unique) subtree with the highest payoff for player i. For all other players $i' \neq i$, let $s_i'^w = \cup_{a \in Act(w)} s_i'^w a$. Then s^w is a unique Nash Equilibrium. If s^w was not a NE, then the subgame $T_w a'$ would not have had the highest payoff for player i, which is a contradiction, and if s^w was not unique then two or more sub-games must have had the same associated payoff, which is also a contradiction.

The above is quite verbose and perhaps slightly 'hand-wavy', but in essence the reasoning comes down to the fact that each player has at each subgame in the game a distinct payoff for a pure strategy, because each leaf has a distinct payoff and the lack of chance nodes preludes mixing (averaging) of payoffs. Therefore, there can only be a single subgame perfect Nash Equilibrium because it must follow the unique best payoff at each and every node. As normal there can be multiple non-subgame perfect Nash Equilibria due to non-credible threats.

2.3 Part (c)

First note that every finite extensive form perfect information game has a Nash Equilibrium in pure strategies[1]. Since all sub-games of such a game are themselves finite extensive form perfect information games, all finite extensive form perfect information games have subgame perfect pure Nash Equilibrium. Therefore, to find a finite extensive form game that contains a pure Nash Equilibrium but not any subgame perfect pure Nash Equilibria, the game must be of imperfect information.

Such a game is given in Figure 3. There are pure Nash Equilibria - one example is given by the blue arrows in the diagram - however there are no subgame perfect pure Nash Equilibria, as it is impossible for Player 2 to decide uniquely what move to make since they may be in either of their possible nodes. There is (of course) a subgame perfect *mixed* Nash

Equilibrium, given by both Player 1 and Player 2 mixing $\frac{1}{2}$ on each of their nodes in the right subgame.

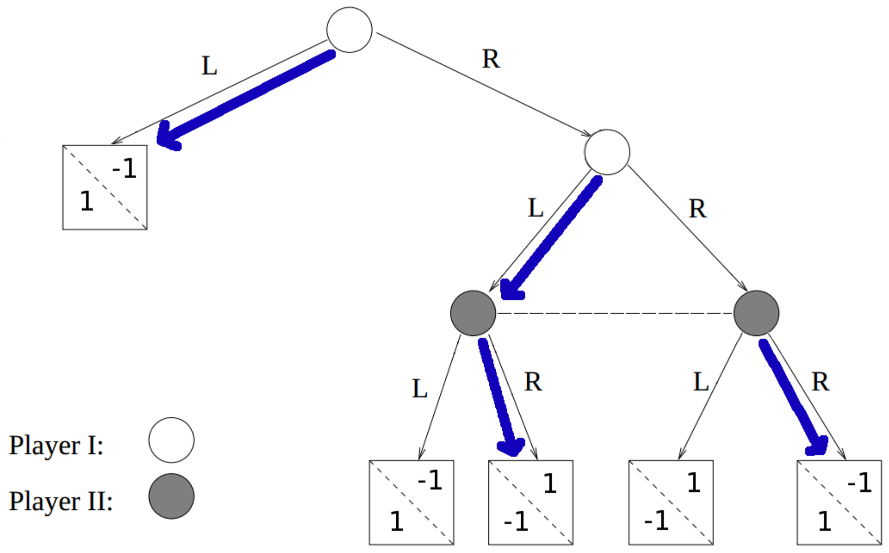

Figure 3: A finite extensive form game with no subgame perfect pure Nash Equilibrium. Information sets are denoted by a dashed line. A non-subgame perfect pure NE is given by the blue arrows.

3 Question 3

The winning condition requires only some overlap between inf(π) and F, i.e. that there exists some $v \in F$ such that v occurs infinitely often in the play π. As such, it clearly suffices to determine if Player 1 can win for each of the vertices in F individually, as if $\inf(\pi) \cap F' \neq \emptyset$ and $F' \subseteq F$, then int(π) must overlap F too.

Next, consider what the requirements are for Player 1 to be able to make sure (using a memoryless strategy) that some vertex $v \in F$ appears on the infinite play. Clearly, there must be some path that Player 1 can force (has a memoryless strategy for) from the starting node v_0 to the goal vertex, as v must be reachable. Additionally, there must also be some path that Player

1 can force (has a memoryless strategy for) from v back to itself, in order to create the infinite play that will contain v an infinite number of times. Convienently, there is already an algorithm to determine if this is true!

Consider the algorithm given in Lecture 12[CITE] for computing memoryless value- achieving strategies for finitistic win-lose games. Setting the 'target' vertex to some $v \in F$ and running the algorithm presents a set Win_1, containing the vertices from which Player 1 can force a path to v. Therefore, for the two conditions outlined above to be true, it simply suffices to check if both $v_0 \in Win_1$ (v is reachable) and that there is some v' such that there is an edge from v to v' and $v' \in Win_1$. Combining this with the need to find only one overlapping vertex gives the following algorithm:

- For $v_{goal} \in F$:
    - Execute the algorithm from the lecture on the graph, starting with $Good = \{v_{goal}\}$ and $St_1 = \{\}$.
    - Check if $v_0 \in Win_1$. If not, move onto the next v_{goal}.
    - Check if $\exists v'$ s.t. $(v_{goal}, v') \in E$ and $v' \in Win_1$. If so, return a win for Player 1, with the strategy $St_1 \cup \{v_{goal} \to v'\}$. Otherwise, move onto the next v_{goal}.
- If no v_{goal} resulted in a win for Player 1, Player 2 wins. Form the the strategy St_2 as follows. For each $v \in pl(2)$:
    - If $\exists (v, v') \in E$ s.t. $v' \notin Win_1$, let $St_2 = St_2 \cup \{v \to v'\}$.

Note that it suffices to form St_2 from the result of the algorithm on the final v_{goal}, as if using St_2 resulted in a win from some $v \in F$, that win would already have been found by one of the iterations of the loop.

With regards to algorithmic complexity, the maximum size of F (and thus the iterations of the loop) is bounded by the number of vertices, n. The inner algorithm is polynomial for some k (I belive that $k = 3$ but it's not important), and so the algorithm is $n * n^k = n^{k+1}$; that is, it has polynomial complexity.

4 Question 4

A pure strategy Nash Equilibrium can be found via Rosenthal[2], by first choosing pure strategies for each player randomly and then iteratively applying strict improvement steps. During this question I will give pure strategies as the paths they cause the players to take through the game, for simplicity.

My initial strategies were:

- Player 1: $S_1 \to B_5 \to B_4 \to T_2 \to S_3 \to T_1$
- Player 2: $S_2 \to B_6 \to B_3 \to B_2 \to B_4 \to T_2$
- Player 3: $S_3 \to B_3 \to T_1 \to B_2 \to B_8 \to T_3$

The cost for these paths for players 1, 2, and 3 are 6, 6, and 5 respectively (as players 1 and 2 share the edge $B_4 \to T_2$). Checking for strict improvements, an immediate and obvious change is for player 1 to move directly from S_1 to B_4, skipping B_5. This reduces player 1s cost to 5. At this point there are no longer any more strict improvement steps - each alternate path for the players has either the same cost or a higher one - and so we have reached a pure Nash Equilbrium:

- Player 1: $S_1 \to B_4 \to T_2 \to S_3 \to T_1$
- Player 2: $S_2 \to B_6 \to B_3 \to B_2 \to B_4 \to T_2$
- Player 3: $S_3 \to B_3 \to T_1 \to B_2 \to B_8 \to T_3$

Interestingly this profile of strategies is far from 'optimal', in that neither the average nor individual costs are as low as they can be. This of course is not surprising as Nash Equilibria do not have to be optimal in any sense, but for interest I enumerated the shortest paths from each S_i to T_i (see Appendix A), and found that there actually exists a Nash Equilibrium that uses shortest paths for each player:

- Player 1: $S_1 \to B_4 \to B_7 \to B_3 \to T_1$
- Player 2: $S_2 \to B_6 \to B_3 \to S_3 \to B_4 \to T_2$
- Player 3: $S_3 \to B_3 \to B_2 \to B_8 \to T_3$

This means that this game represents the relatively rare case where (an arrangement of) the shortest paths actually gives a Nash Equilibrium!

5 Question 5

5.1 Part (a)

The VCG mechanism for the auction is similar to that presented in the course, with the differential that the possible 'candidates' or outcomes of the auction are now replaced by vectors representing 'feasible' allocations of books; $S = (S_1, ..., S_n)$. A vector S is feasible if it is possible to give the allocations specified to the buyers, i.e. if for all $i \neq j$, $S_i \cap S_j = \emptyset$. The mechanism then computes the optimal outcome:

$$S^* = (S_1^*, ..., S_n^*) = \arg\max_S \sum_{i=1}^{n} v_i'(S_i)$$

Each buyer is then charged a price p_i that depends on the 'damage' they do to other buyers:

$$p_i = \left(\max_S \sum_{j \neq i} v_j'(S_j) \right) - \sum_{j \neq i} v_j'(S_j^*)$$

The reason that this causes truthfulness in the buyers is essentially identical to the one presented in the lecture.

In summary, the rules of the auction are:

- Each player submits a valuation function $v_i' : 2^D \to \mathbb{R}_0^+$, which defines a bid for every subset of books $S \subseteq D$.
- The feasible allocation sets are enumerated: $\{S_i\}_{i=1}^n$, such that if $i \neq j$, $S_i \cap S_j = \emptyset$.
- An allocation $(S_1^*, ..., S_n^*)$ is chosen that maximizes $\sum_{i=1}^n v_i'(S_i)$.
- Each buyer i is charged $p_i = (\max_S \sum_{j \neq i} v_j'(S_j) - \sum_{j \neq i} v_j'(S_j^*)$.

5.2 Part (b)

Computationally speaking the algorithm is infeasible. There are exponentially many feasible sets, and therefore finding the optimal set S^* is NP-hard.

A Code for Shortest Path Finding (Question 4)

```
class Node(object):

  def __init__(self, label):
    self.label = label
    self.outs = set()

  def add_edge(self, other_node):
    self.outs.add(other_node)

  def __eq__(self, other):
    return self.label == other.label

  def __ne__(self, other):
    return not self.__eq__(other)

  def __hash__(self):
    return hash(self.label)


def dfs(current_node, target_node, path_so_far, current_depth, max_depth):
  path_copy = list(path_so_far)
  path_copy.append(current_node.label)

  if current_node == target_node:
    print ' -> '.join(path_copy)
    return

  if current_depth == max_depth:
    return

  for child_node in current_node.outs:
    dfs(child_node, target_node, path_copy, current_depth + 1, max_depth)


def find_paths(from_node, to_node, max_depth):
  dfs(from_node, to_node, [], 0, max_depth)
```

```
def main():
  s1 = Node("s1")
  t1 = Node("t1")

  s2 = Node("s2")
  t2 = Node("t2")

  s3 = Node("s3")
  t3 = Node("t3")

  b1 = Node("b1")
  b2 = Node("b2")
  b3 = Node("b3")
  b4 = Node("b4")
  b5 = Node("b5")
  b6 = Node("b6")
  b7 = Node("b7")
  b8 = Node("b8")

  s1.add_edge(t3)
  s1.add_edge(b4)
  s1.add_edge(b5)

  t1.add_edge(s3)
  t1.add_edge(b2)

  s2.add_edge(b6)
  s2.add_edge(b8)

  t2.add_edge(s3)
  t2.add_edge(b1)

  s3.add_edge(t1)
  s3.add_edge(b3)
  s3.add_edge(b4)

  t3.add_edge(s1)
  t3.add_edge(s2)
  t3.add_edge(b1)
  b1.add_edge(b2)
  b1.add_edge(b5)

  b2.add_edge(b3)
  b2.add_edge(b4)
  b2.add_edge(b6)
  b2.add_edge(b8)

  b3.add_edge(s3)
  b3.add_edge(t1)
  b3.add_edge(b2)

  b4.add_edge(t2)
  b4.add_edge(b7)

  b5.add_edge(b4)

  b6.add_edge(b3)
  b6.add_edge(s2)

  b7.add_edge(b3)

  b8.add_edge(t3)
  b8.add_edge(b1)

  print "Paths from s1 to t1:"
  find_paths(s1, t1, 6)
  print

  print "Paths from s2 to t2:"
  find_paths(s2, t2, 6)
  print

  print "Paths from s1 to t1:"
  find_paths(s3, t3, 6)
  print


if __name__ == "__main__":
  main()
```

References

[1] Kuhn, Harold W. *Extensive Games and the Problem of Information.* Contributions to the Theory of Games 2.28 (1953).

[2] R.W. Rosenthal. *The Network Equilibrium Problem in Integers*, Networks (1973).