Samuel Singh – 1024640 **Homework 1**

MAI 5100 **2025-04-27**

Part 2

Question 1

R&N Problem 5.12: (8 points) Describe how the minimax and alpha–beta algorithms change for two-player, non-zero-sum games in which each player has a distinct utility function and both utility functions are known to both players. If there are no constraints on the two terminal utilities, is it possible for any node to be pruned by alpha–beta? What if the player's utility functions on any state differ by at most a constant k, making the game almost cooperative?

The current implementation of the minimax and alphabeta algorithms assumes that the game is a zero-sum ie. a gain for one agent is a loss for the other agent and vice-versa. The pacman agent acts as the maximizer and so attempts to maximize the score while the ghost acts as the minimizer, attempting to reduce the score. This is suited for adversarial games where all agents are in direct competition with one another, however, when dealing with non zero sum games, the assumptions made shift considerably. Each agent possesses a distinct utility, eg. the pacman might aim to maximize the score while the ghosts might prioritize staying alive. Not all agents are strictly adversarial as some of their objectives might be conflicting, independent, and maybe even cooperative in specific conditions.

To modify the zero sum algorithm for multiple players, a few changes must be made. The first one is the metric for measuring utility. In a zero sum environment a scalar quantity is used ie the score. For a non zero sum environment a vector quantity is used eg (pacmanUtility, ghostUtility).

Maximization would need to be done on individual objectives where each agent would select actions that maximize their own component of the utility vector ie. Pacman agents would maximize the first element, while the ghosts would maximize the second element (or their respective index in the multiagent setup). This means that the ghosts no longer attempt to directly minimize the pacman utility, but rather act 'selfishly' and select actions that are 'best' for their individual utility function. During tree traversal, each agent evaluates and selects actions based solely on their own utility outcomes. The concept of a universal "best move" is replaced by what is best from each agent's perspective, fundamentally changing how decisions must be evaluated in multi-agent contexts.

When adapting Alpha-Beta pruning to non-zero-sum games, significant challenges arise due to the incompatibility with single thresholds. Alpha-Beta pruning in its classic form relies on comparing single scalar values (e.g., Pacman's score), but in a non-zero-sum game, decisions depend on multiple utility values, one for each agent. This creates a lack of safe pruning conditions, as it is not safe to prune based on one agent's utility alone. A move that appears suboptimal for one agent (e.g., Pacman) may be highly favorable for another (e.g., a ghost).

This situation necessitates multi-dimensional thresholds where alpha and beta must be represented as tuples or vectors. For example, alpha would become (alphaPacman, alphaGhost). Pruning becomes viable only if a potential branch is dominated across all utility dimensions, which is a rare occurrence unless the utility functions are closely aligned. Additionally, determining when to prune requires Pareto dominance checks or multi-criteria decision analysis, which significantly increases computational complexity.

A special case exists when utilities differ by at most a constant value k, where the game may be considered almost cooperative. The partial alignment of goals implies that actions which are good for one agent are not drastically worse for the other. This allows for approximated pruning, using the assumption that large disparities in utility indicate dominance. If Action A results in utilities significantly higher (by more than k) for both agents compared to Action B, then Action B can be safely pruned. This approximation enables a heuristic form of alpha-beta pruning, which may not be exact but still offers computational efficiency without major loss of decision quality.

Question 2

R&N Problem 6.8: (8 points) Consider the graph with 8 nodes A1,A2,A3,A4,H,T,F1,F2 . Ai is connected to Ai+1 for all i, each Ai is connected to H, H is connected to T, and T is connected to each Fi . Find a 3-coloring of this graph by hand using the following strategy: backtracking with conflict-directed back jumping, the variable order A1,H,A4,F1,A2,F2,A3,T , and the value order R,G,B.

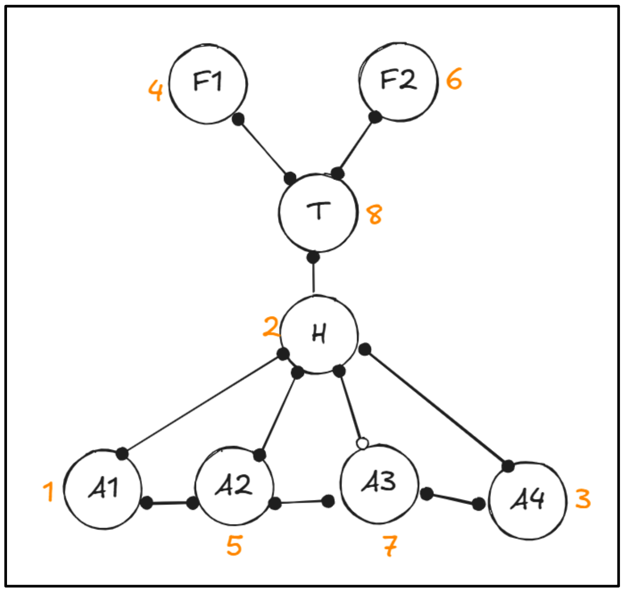

Figure 1 – R&N problem

1. Since the color sequence is RGB, we initialize by setting A1 = R and move to the next node, H.
2. Set H = R but this conflicts A1 so increment the RGB.
3. Set H = G and move to the next node, A4.
4. Set A4 = R (start over RGB)
5. Set F1 = R (start over RGB)
6. Set A2 = R but this will conflict with A1. Incrementing A2 to G but this will now conflict H. Incrementing A2 to B.
7. Set F2 = R.
8. Set A3 = R but this conflicts with A4. Setting A3 to G conflicts with H. Incrementing A3 to B conflicts with A2. At this point the A3 node is in conflict with A2, A4 and H. At this point based on node placement, the most recent node is F2 but this has no impact on the current conflict so moving further back we get to node A2.

9. At this point the nodes under contention with A2(B) are A1(R), H(G) and A4(R). Any color change on A2 at this point would result in a conflict.
10. Moving to the next most recent node is F1 but this has no impact on the conflict so move to the next most recent node, A4. Set A4 = G will conflict with H. Set A4 = B.
11. Moving forward again, the next node is F1. Set F1 = R
12. Set A2 = R but this conflicts with A1. Set A2 = G but this conflicts H. Set A2 = B.
13. Set F2 = R
14. Set A3 = R
15. Set T = R but this conflicts with both F1 and F2. Set T = G but this conflicts with H. Set T = B.

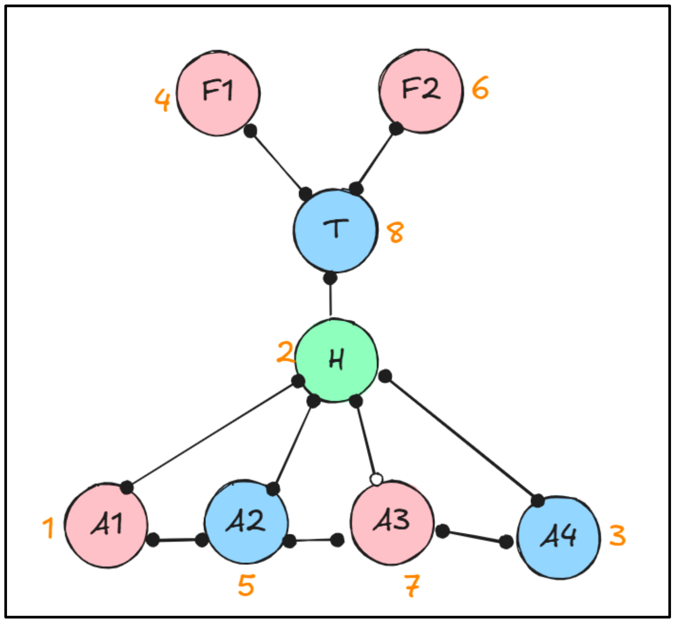

Figure 2 - R&N problem solution

Question 3

Course Scheduling (14 points) You are in charge of scheduling for computer science classes that meet Mondays, Wednesdays and Fridays. There are 5 classes that meet on these days and 3 professors who will be teaching these classes. You are constrained by the fact that each professor can only teach one class at a time.

The classes are:

1. Class 1 - Intro to Programming: meets from 8:00-9:00am
2. Class 2 - Intro to Artificial Intelligence: meets from 8:30-9:30am
3. Class 3 - Natural Language Processing: meets from 9:00-10:00am
4. Class 4 - Computer Vision: meets from 9:00-10:00am
5. Class 5 - Machine Learning: meets from 10:30-11:30am

The professors are:

1. Professor A, who is qualified to teach Classes 1, 2, and 5.
2. Professor B, who is qualified to teach Classes 3, 4, and 5.
3. Professor C, who is qualified to teach Classes 1, 3, and 4.

1. Formulate this problem as a CSP problem in which there is one variable per class, stating the domains (after enforcing unary constraints), and binary constraints. Constraints should be specified formally and precisely, but may be implicit rather than explicit.
2. Draw the constraint graph associated with your CSP.

The node definition as used in the constraint graph is given in the table below:

Node	Definition
1	Class 1 - Intro to Programming
2	Class 2 - Intro to Artificial Intelligence
3	Class 3 - Natural Language Processing
4	Class 4 - Computer Vision
5	Class 5 - Machine Learning

Table 1 – Node definition

The unary constraint when applied to each professor is defined in the table below:

Professor	Unary Constraint (Classes)
A	1,2,5
B	3,4,5
C	1,3,4

Table 2 – Unary Constraint

Based on the table above each node (class) can be assigned possible professors as expressed in the table below:

Node (Class)	Time	Professor (domain)
1	8:00-9:00	A,C
2	8:30-9:30	A
3	9:00-10:00	B,C
4	9:00-10:00	B,C
5	10:30-11:30	A,B

Table 3 – Professor Assignment

This is represented graphically below. Since there is no overlap for the fifth class (node 5) it is disconnected from the rest of the nodes as it has no influence on the constraint satisfaction problem.

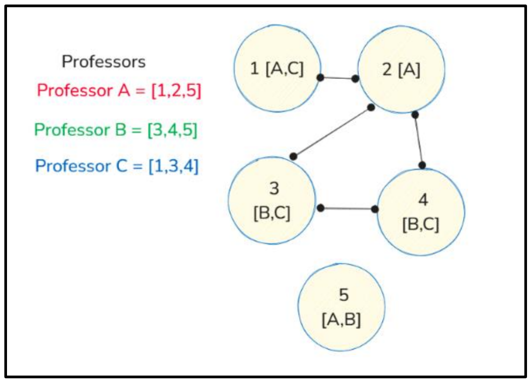

Figure 3 - Professor Assignment

For any class that overlap then: assignment(x) ≠ assignment(y). This means that if there is a conflict where any of the class times overlap with each other, then that class must be assigned a different professor based on the list of available qualified professors for that class. From the table 4 below, it was found that nodes 1&2, 2&3, 2&4 and 3&4 must have different professors.

Node Pair	**Overlap**
1,2	*Yes*
1,3	No
1,4	No
1,4	No
2,3	*Yes*
2,4	*Yes*
2,5	No
3,4	*Yes*
3,5	No
4,5	No

Table 4 – Node Pair Conflicts

This means that the binary constraints are such that the nodes(classes):

- 1≠2
- 2≠3
- 2≠4
- 3≠4

Using this, a node exploration tree was developed to derive the final solution. This graph is presented in figure 4 below. The final constraint graph is presented in the figure 5 below and the final solution is tabulated in the table 5 below.

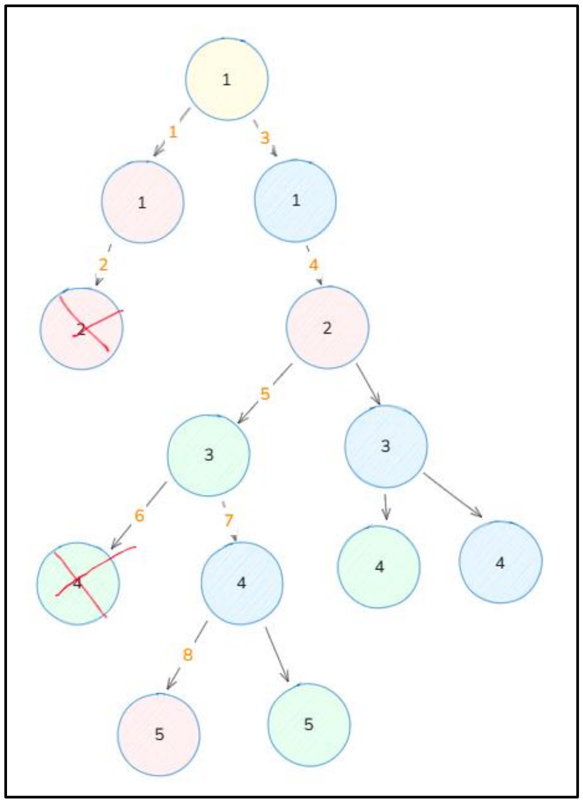

Figure 4 - Node exploration tree

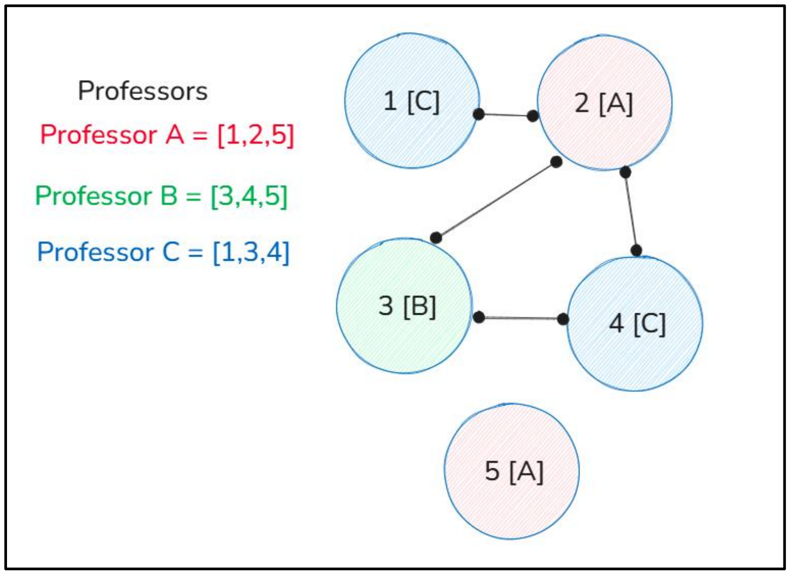

Figure 5 - Constraint Graph

Class	Time	Professor
1	8:00-9:00	C
2	8:30-9:30	A
3	9:00-10:00	B
4	9:00-10:00	C
5	10:30-11:30	A

Table 5 – Final Schedule.

Based on the table above, the Professor C will teach the class 1, Professor A will teach class 2, Professor B will teach class 3, professor C will teach class 4 and professor A will teach class 5. Based on the constraint graph and the nodal exploration tree, it can be noted that class 3 and 4 could have their professors interchanged, ie. class 3 could be taught by professor C and class 4 could be taught by professor B. Similarly, since node 5 (class 5) is detached from the rest of the graph, the lecturers qualified for that class could be interchanged therefore professor B could also teach class 5 without creating any further conflicts.