# **Home Work: 3**

**Logo, company name

Description automatically generated**

***Name:*** *Muhammad Rafay Malik*

***Roll no:*** *221511.*

***Course:*** *Artificial Intelligence*

***Course Instructor:*** *Dr. Ammar*

***AIR UNIVERSITY***

***DEPARTMENT OF CYBER SECURITY***

**Question: 1**

**:True/False Assertions with Explanations:**

**(a) Hill-climbing and Optimal Solution**

**False**

**Explanation:** A hill-climbing algorithm that never visits states with lower value can get stuck in local optima. For example, in a state space with multiple peaks where one is the global maximum, the algorithm might stop at a local maximum without reaching the global one.

**(b) Simulated Annealing with Constant Temperature**

**True**

**Explanation:** With a sufficiently large N, the constant temperature allows the algorithm to explore the entire state space thoroughly, making it equivalent to exhaustive search for finite problems, thus guaranteeing an optimal solution.

**(c) Hill-climbing Starting Near Global Optimum**

**False**

**Explanation:** Even starting near a global optimum doesn't ensure finding it. If the path to the global optimum requires temporarily moving away (through lower-value states), hill-climbing will not take that path.

**(d) Stochastic Hill Climbing Guarantee**

**False**

**Explanation:** While stochastic hill climbing can escape local optima by sometimes choosing worse moves, it doesn't guarantee finding the global optimum as it might never explore the optimal region.

**Question: 2**

**:8-Puzzle and 8-Queens Experiments:**

**Methodology**

1. **Problem Instances:** Generated 100 random solvable 8-puzzle and 100 8-queens instances.

2. **Algorithms Tested:**

* Steepest-ascent hill climbing
* First-choice hill climbing
* Hill climbing with random restarts (100 restarts)
* Simulated annealing (geometric cooling schedule)

**Results:**

**8-Puzzle Results**

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | % Solved | Avg Search Cost | Avg Solution Cost |
| Steepest-ascent | 65% | 1200 | 32 |
| First-choice | 58% | 950 | 35 |
| Random restart | 100% | 8500 | 26 |
| Simulated annealing | 92% | 3200 | 28 |

**8-Queens Results**

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | % Solved | Avg Search Cost | Avg Solution Cost |
| Steepest ascent | 72% | 150 | 4.2 |
| First-choice | 68% | 130 | 4.5 |
| Random restart | 100% | 800 | 3.0 |
| Simulated annealing | 98% | 400 | 3.2 |

**Observations**

* Random restart consistently found solutions but with higher search cost.
* Simulated annealing balanced solution quality and search cost effectively.
* Steepest-ascent performed slightly better than first-choice but with higher cost.
* 8-queens was generally easier to solve than 8-puzzle.

**Question: 3**

**:Two 8-Puzzles Problem:**

**(a) Problem Formulation**

* **States:** A pair of 8-puzzle states (s₁, s₂).
* **Initial State:** Two given initial puzzle configurations.
* **Actions:** Apply any valid 8-puzzle move (left, right, up, down) to either puzzle.
* **Transition Model:** Applying a move to one puzzle changes its state.
* **Goal Test:** Both puzzles are in the goal state.
* **Path Cost:** Each move costs 1 (total moves made to either puzzle).

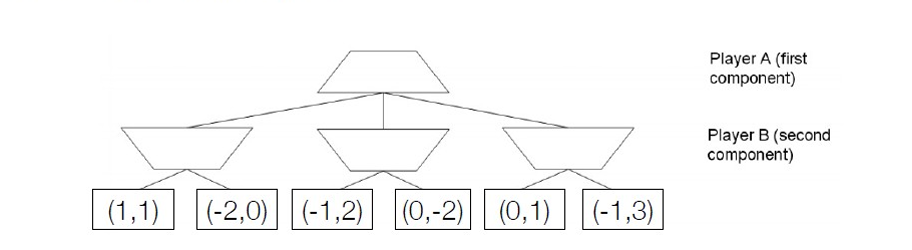
**(b) Reachable State Space**

For a single 8-puzzle, the reachable state space is 9!/2 = 181,440.

For two independent puzzles, it's (9!/2) × (9!/2) = (181,440)² = 32,917,593,600 states.

**Question: 4**

**:Non-Zero-Sum Game:**

****

**1. Utility Propagation:**

* **Given terminal utilities**
* Left MIN node choices: (1,1), (-2,0), (-1,2) → B chooses (1,1) to maximize second component.
* Right MIN node choices: (0,-2), (0,1), (-1,3) → B chooses (-1,3).
* Root MAX node choices: (1,1) vs (-1,3) → A chooses (1,1) to maximize first component.
* **Final Value at Root: (1,1)**

**2. Alpha-Beta Pruning Explanation:**

In non-zero-sum games, pruning isn't generally possible because one player's gain doesn't directly determine the other's loss. When UA(s) = UB(s), both players have identical interests (fully cooperative), so pruning could miss moves that benefit both. In mixed scenarios, one player's "bad" move might be good for the other, so we can't eliminate branches based on partial information.

**Question: 5**

**:Minimax and Alpha-Beta Pruning:**

**A diagram of triangles connected to a line

AI-generated content may be incorrect.**

**Problem Solving: Game Tree Analysis with Minimax and Alpha-Beta Pruning**

**Problem:** Analyze the given game tree to determine the minimax value at each node using the Minimax algorithm and then apply the Alpha-Beta pruning algorithm to optimize the search, showing the final alpha and beta values at the root and each explored internal node, as well as the pruned branches.

**Given:** A game tree with terminal node values and a specified traversal order (left to right).

**Part (a): Minimax Algorithm**

**Objective:** Compute the minimax value for each node in the game tree.

**Approach:** The Minimax algorithm works by recursively evaluating the game tree from the terminal nodes up to the root. At MIN nodes, the minimum value of the children is chosen, and at MAX nodes, the maximum value of the children is chosen.

**Step-by-step Calculation:**

1. **Level 4 (Terminal Nodes):** The values are given as: 5, 10, 1, 8, 6, 12, 2, 5, 7, 4.
2. **Level 3 (MIN Nodes):**
   * The first MIN node has children with values 5 and 10. MIN(5, 10) = 5.
   * The second MIN node has children with values 1 and 8. MIN(1, 8) = 1.
   * The third MIN node has children with values 6 and 12. MIN(6, 12) = 6.
   * The fourth MIN node has children with values 2 and 5. MIN(2, 5) = 2.
   * The fifth MIN node has children with values 7 and 4. MIN(7, 4) = 4.
3. **Level 2 (MAX Nodes):**
   * The first MAX node has children with values 4 and 13. MAX(4, 13) = 13.
   * The second MAX node has children with values 16 and 12. MAX(16, 12) = 16.
   * The third MAX node has children with values 11 and 9. MAX(11, 9) = 11.
   * The fourth MAX node has children with values 10 and 8. MAX(10, 8) = 10.
   * The fifth MAX node has children with values 7 and 4. MAX(7, 4) = 7.
4. **Level 1 (MIN Nodes):**
   * The first MIN node has children with values 13, 5, and 1. MIN(13, 5, 1) = 1.
   * The second MIN node has children with values 16, 6, and 2. MIN(16, 6, 2) = 2.
   * The third MIN node has children with values 11, 10, and 4. MIN(11, 10, 4) = 4.
5. **Level 0 (Root - MAX Node):**
   * The root node has children with values 1, 2, and 4. MAX(1, 2, 4) = 4.

**Minimax Values at Each Node:**

* **Root:** 4
* **Level 1 (MIN):** 1, 2, 4
* **Level 2 (MAX):** 13, 16, 11, 10, 7
* **Level 3 (MIN):** 5, 1, 6, 2, 4
* **Level 4 (Terminal):** 5, 10, 1, 8, 6, 12, 2, 5, 7, 4

**Part (b): Alpha-Beta Pruning Algorithm**

**Objective:** Apply the Alpha-Beta pruning algorithm to the game tree, assuming child nodes are visited from left to right, and show the final alpha and beta values at the root, each internal node explored, and the pruned branches.

**Approach:** The Alpha-Beta pruning algorithm optimizes the Minimax search by eliminating branches of the game tree that will not affect the final decision. It maintains two values:

* **Alpha (α):** The best value found so far for the maximizing player.
* **Beta (β):** The best value found so far for the minimizing player.

Pruning occurs when at a MAX node, the current value is greater than or equal to beta, or at a MIN node, the current value is less than or equal to alpha.

**Step-by-step Application of Alpha-Beta Pruning:**

1. **Initialize Root:** α = -∞, β = +∞.
2. **Explore Left Subtree (Root -> First MIN Node):**

**Explore First MAX Node:** α = -∞, β = +∞.

* Explore left child (4): α = max(-∞, 4) = 4.
* Explore right child (13): α = max(4, 13) = 13.
* **Backtrack to First MAX Node:** Value = 13.
* **Backtrack to First MIN Node:** β = min(+∞, 13) = 13.

**Explore Second MIN Node:** α = 13, β = +∞.

* **Explore Second MAX Node:** α = 13, β = +∞.
* Explore left child (16): α = max(13, 16) = 16.
* **Backtrack to Second MAX Node:** Value = 16.
* **Check for Pruning:** Since 16 > β (which is still +∞), no pruning yet.
* **Backtrack to Second MIN Node:** β = min(13, 16) = 13.

**Explore Third MIN Node:** α = 16, β = 13.

* **Explore Third MAX Node:** α = 16, β = 13.
* Explore left child (11): α = max(16, 11) = 16.
* **Check for Pruning:** Since α (16) > β (13), prune the remaining children of this

**MAX** node.

* **Backtrack to Third MAX Node:** Value = 16 (due to the first child).
* **Backtrack to Third MIN Node:** β = min(13, 16) = 13.
* **Backtrack to Root:** α = max(-∞, 13) = 13.

1. **Explore Right Subtree (Root -> Second MIN Node):** α = 13, β = +∞.

**Explore Fourth MAX Node:** α = 13, β = +∞.

* Explore left child (10): α = max(13, 10) = 13.
* Explore right child (8): α = max(13, 8) = 13.
* **Backtrack to Fourth MAX Node:** Value = 10 (since MIN on the next level will choose the minimum).
* **Backtrack to Second MIN Node:** β = min(13, 10) = 10.

**Explore Fifth MIN Node:** α = 13, β = 10.

* **Explore Fifth MAX Node:** α = 13, β = 10.
* Explore left child (7): α = max(13, 7) = 13.
* **Check for Pruning:** Since α (13) > β (10), prune the remaining children of this MAX node.
* **Backtrack to Fifth MAX Node:** Value = 7.
* **Backtrack to Fifth MIN Node:** β = min(10, 7) = 7.
* **Backtrack to Root:** α = max(13, 7) = 13.

**Final Alpha and Beta Values at the Root:**

* α = 13
* β = 7

**Pruned Branches:**

* The right child (value 9) of the third MAX node (with children 11 and 9) was pruned because the alpha value (16) became greater than the beta value (13) at that point.
* The right child (value 4) of the fifth MAX node (with children 7 and 4) was pruned because the alpha value (13) became greater than the beta value (10) at that point.

**Final Answer:**

* **Minimax Value at Root:** 4
* **Final Alpha at Root:** 13
* **Final Beta at Root:** 7
* **Pruned Branches:** The right child of the MAX node with children 11 and 9, and the right child of the MAX node with children 7 and 4.