Introduction to Artificial Intelligence - Final Project

Le Dang Nguyen
Faculty of Information Technology,
Ton Duc Thang University,
Ho Chi Minh City, Vietnam,
522K0020@student.tdtu.edu.vn

Pai Hein Kyaw
Faculty of Information Technology,
Ton Duc Thang University,
Ho Chi Minh City, Vietnam,
523K0078@student.tdtu.edu.vn

Thiha Aung
Faculty of Information Technology,
Ton Duc Thang University,
Ho Chi Minh City, Vietnam,
523K0073@student.tdtu.edu.vn

Thin Lei Sandi
Faculty of Information Technology,
Ton Duc Thang University,
Ho Chi Minh City, Vietnam,
523K0075@student.tdtu.edu.vn

Msc. Nguyen Thanh An
Faculty of Information Technology,
Ton Duc Thang University,
Ho Chi Minh City, Vietnam,
nguyenthanhan@tdtu.edu.vn

***Abstract*—This report presents implementations of four tasks for an Introduction to AI course. We developed a Simulated Annealing algorithm with a custom temperature schedule to optimize a 3D function, a 9×9 Tic-Tac-Toe game with a heuristic alpha-beta pruning AI opponent, a constraint satisfaction solver for cell coloring puzzles using propositional logic and Glucose3 SAT solver, and a Naïve Bayesian Classifier for predicting student outcomes from discretized quiz scores.**

***Index Terms*—simulated annealing, alpha-beta pruning, constraint satisfaction, SAT solving, Naïve Bayesian classification.**

I. Task 1: Simulated Annealing Search on 3D Surfaces

A. Introduction and Problem Statement

Simulated Annealing (SA) provides a metaheuristic optimization approach capable of escaping local extrema and implements SA to find the maximum value of the function:

$$f(x,y) = \sin\left(\frac{x}{8}\right) + \cos\left(\frac{y}{4}\right) - \sin\left(\frac{x \cdot y}{16}\right) + \cos\left(\frac{x^2}{16}\right) + \sin\left(\frac{y^2}{8}\right) \tag{1}$$

The implementation addresses three objectives: developing a Simulated Annealing algorithm starting from the origin $(0, 0)$ with a step size of $\frac{\pi}{32}$, designing an efficient temperature schedule function.

B. Theoretical Background

*1) **Simulated Annealing (SA)**:* SA maintains a current state with value E(s) while generating neighbor states with value E(s'). The algorithm accepts improvements unconditionally, but also accepts suboptimal moves with probability:

$$P(s \to s') = \exp\left(\frac{E(s') - E(s)}{T}\right) \tag{2}$$

where T represents temperature. The temperature decreases according to a cooling schedule T(t)

C. Algorithm Design

The algorithm begins at the origin (0,0) and iteratively explores by taking random steps of size /32.

Algorithm 1 Simulated Annealing Search

```
Initialize current ← (0,0), best ← current, evaluate values
for t = 1 to max_iterations do
    temperature ← T(t), generate neighbor with step π/32
    neighbor_value ← f(neighbor)
    if neighbor_value > current_value or rand() < exp((neighbor_value − current_value)/temperature) then
        current ← neighbor, update current_value
        if neighbor_value > best_value then
            best ← neighbor, update best_value
    Record current in search path
return best, best_value
```

Temperature Schedule: The algorithm uses a novel three-phase adaptive temperature schedule to guide the process:

$$T(t) = \begin{cases} T_0 \cdot 0.998^t & \text{if } t < 0.1t_{\max} \\ T_0 \cdot 0.995^t & \text{if } 0.1t_{\max} \le t < 0.5t_{\max} \\ T_0 \cdot 0.99^t & \text{if } t \ge 0.5t_{\max} \end{cases} \tag{3}$$

This schedule defines three distinct search phases:

- **Early exploration (first 10%):** A slow cooling rate (0.998^t), enabling broad exploration.
- **Transition phase (next 40%):** A moderate cooling rate (0.995^t) balances exploration and refinement.
- **Exploitation (final 50%):** A faster cooling rate (0.99^t) limits the acceptance of worse solutions and promotes convergence.

The implementation employs a object-oriented structure:

- **Function3D class:** Handles function definition, evaluation, and surface visualization.
- **SimulatedAnnealing class:** Contains the optimization logic.

D. Experimental Results

1) **Optimization Performance**:
   - The algorithm consistently identified a best state around $(x, y) \approx (3.33, -3.53)$ with $f(x, y) \approx 3.479$, near the global maximum.

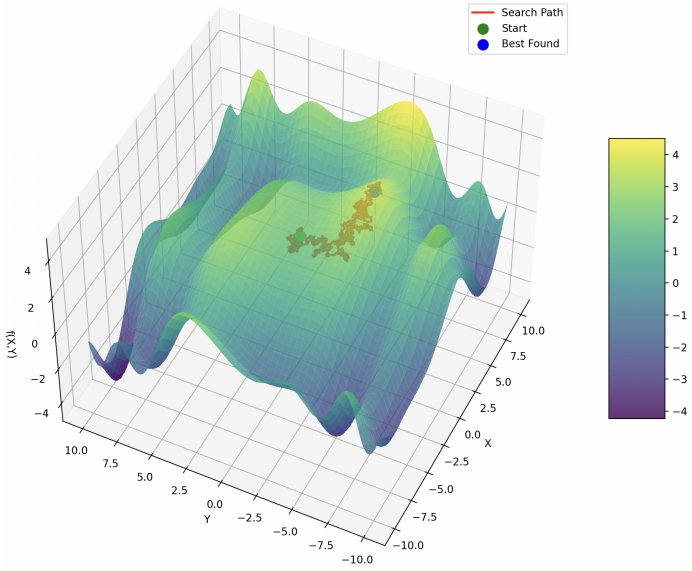

Fig. 1. 3D surface plot of $f(x, y)$ with the SA search path (red line), starting point (green marker), and best-found state (blue marker).

2) **Convergence Behavior**: Function value plots showed rapid exploration in the first 1,500 iterations, jumps to escape local maxima. The temperature schedule plot confirmed the three-phase cooling's effectiveness, with early exploration and later convergence.

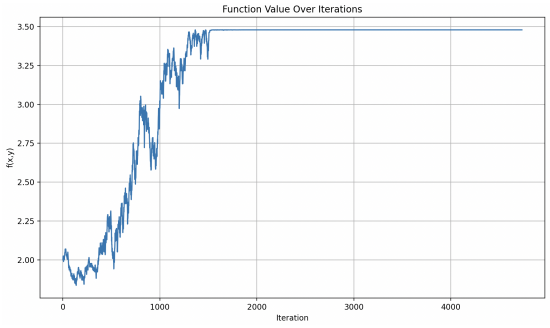

Fig. 2. Function value over iterations, showing the SA algorithm's convergence behavior.

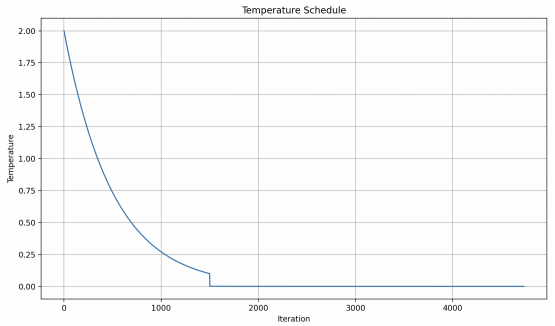

Fig. 3. Temperature schedule over iterations, illustrating the three-phase cooling strategy.

II. Task 2: 9x9 Tic-Tac-Toe with Heuristic Alpha-Beta Search

A. Introduction and Problem Statement

This section explain how a 9x9 Tic-Tac-Toe game was implemented a computer opponent using heuristic alpha-beta search, a combination of traditional alpha-beta pruning with h-minimax techniques

B. Theorical Background

1) Heuristic Alpha-Beta Search: Alpha-Beta Pruning is an optimization technique for the minimax algorithm that reduces the number of nodes evaluated in the search tree.

The algorithm integrates three core components:

1) **Alpha-Beta Pruning**: Eliminates branches, reducing the effective branching factor from b to $\sqrt{b}$.
2) **Depth-Limited Search**: Cuts off search at a predetermined depth L, reducing time complexity from $O(b^d)$ to $O(b^L)$.
3) **Heuristic Evaluation**: Estimates the value of non-terminal positions using domain-specific knowledge.

The heuristic alpha-beta search can be formally defined as:

$$\text{HeuristicMinimax}(s, d, \alpha, \beta) = \begin{cases} \text{Utility}(s) & \text{if } s \text{ is terminal} \\ \text{Eval}(s) & \text{if } d = 0 \\ \max_a \text{HeuristicMinimax}(\text{Result}(s, a), d-1, \alpha, \beta) & \text{if MAX player} \\ \min_a \text{HeuristicMinimax}(\text{Result}(s, a), d-1, \alpha, \beta) & \text{if MIN player} \end{cases}$$

Algorithm 2 ALPHA-BETA SEARCH with H-MINIMAX

```
function HeuristicAlphaBeta(board, depth, isMax, α, β)
    utilityVal ← Utility(board, depth)
    if utilityVal ≠ null then return utilityVal
    if depth = 0 then return EvaluateBoard(board)
    positions ← GetEmptyPositions(board)
    Sort positions by PositionValue()
    best ← isMax ? −∞ : +∞
    for all (row, col) ∈ positions do
        boardCopy ← GetCopy(board)
        MakeMove(boardCopy, row, col, isMax ? playerSymbol : opponentSymbol)
        eval ← HeuristicAlphaBeta(boardCopy, depth-1, !isMax, α, β)
        best ← isMax ? max(best, eval) : min(best, eval)
        if isMax then
            α ← max(α, best)
        else
            β ← min(β, best)
        if β ≤ α then break
    return best
```

With effective pruning, alpha-beta can explore search trees roughly twice as deep as pure minimax.

C. Approach Design

After extensive analysis and testing, we designed a multi-faceted heuristic that considers three primary strategic elements.

1) Class Design for Heuristic Evaluation: Figure 1 shows the class structure for the heuristic evaluation function.

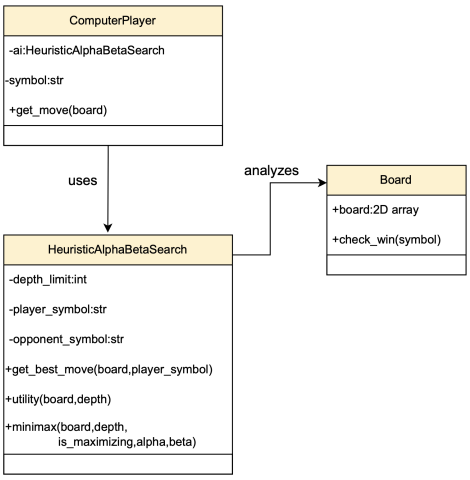

Fig. 4. Class Diagram Showing the Components of the Heuristic Evaluation

The most critical component of our implementation is the heuristic evaluation function.

2) Game Board Visualization: The game features a console-based visualization system that provides and informative interface for players. Figure 5 shows an example of the game board visualization.

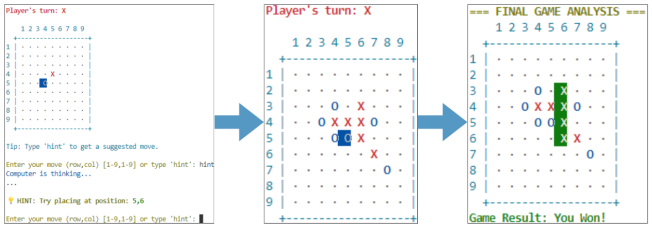

Fig. 5. Visualizatoin of 9x9 Tic-Tac-Toe Game

The game board is displayed as a 9×9 grid with the following features:

- Row and column indices (1-9) for easier coordinate input,
- Player symbols: **'X'** for human (highlighted in red) and **'O'** for computer (highlighted in blue),
- Empty positions marked with dot characters **('·')**.

D. Conclusion

The implemented 9x9 Tic-Tac-Toe with heuristic alpha-beta search achieves:

1) An intelligent computer opponent balancing offensive and defensive play,
2) Quick response times (typically 1-2 seconds),

This implementation demonstrates the effectiveness of combining alpha-beta pruning with h-minimax for games with larger search spaces, showing how a well-designed heuristic function can compensate for limited search depth.

III. Constraint Satisfaction Problems with Propositional Logic

A. Introduction and Problem Statement

This work investigates a spatial coloring puzzle that transforms visual reasoning into computation, involving an m×n grid where cells receive binary red or green assignments governed by numerical constraints required green neighbors.

B. Theorical Background

The cell coloring puzzle is a CSP where each cell is a binary variable, constraints are defined by numerical values in the grid.

- **Propositional Variables**: Each cell (i, j) in the grid is assigned a propositional variable $x_{i,j}$, where $x_{i,j} = 1$ if the cell is green and $x_{i,j} = 0$ if red.
- **Constraints**: For a cell at (i, j) with value n, the number of green cells in its 3x3 neighborhood (including itself) must equal n.
- **SAT Formulation**: The puzzle is translated into a conjunctive normal form formula. A clause is a disjunction of literals and the formula is a conjunction of clauses.
- **Exactly-n Encoding**: For a set of variables, the "exactly-n" constraint is encoded by combining "at-most-n" and "at-least-n" constraints.

C. Approach Design

1) Puzzle Representation: The grid uses a matrix of cells storing integer values (or None for blanks) and boolean color flags. The `ColoringPuzzle` class manages initialization, neighborhood computation, and visualization.

2) Constraint Encoding: The `PropositionalLogicSolver` class encodes the puzzle as a SAT problem:

- **Variable Initialization**: Assigns a unique propositional variable to each cell.
- **Clause Generation**: For each cell with value n, generates CNF clauses ensuring exactly n green cells in its 3x3 neighborhood. The "exactly-n" constraint is implemented by combining "at-most-n" and "at-least-n" clauses.
- **SAT Solving**: Uses the Glucose3 solver to find a satisfying assignment.

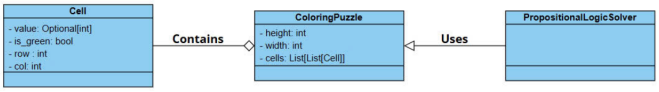

Fig. 6. Class Diagram for Task 3

Algorithm 3 Solve Cell Coloring Puzzle

Input: Matrix M, **Output:** Valid coloring or no solution
Initialize puzzle and assign variables $x_{i,j}$ to each cell
for each numbered cell (i, j) with value n **do**
    Generate CNF clauses: exactly n green cells in 3×3 neighborhood
Solve CNF using Glucose3
if solution found **then**
    Update colors, verify constraints, return solution
else
    Return No solution

Algorithm 4 Exactly-n Constraint Encoding

Input: List of variables V, integer n
Output: List of CNF clauses
if $n = 0$ **then**
    Return Clauses $[[-v]$ for $v \in V]$
if $n = |V|$ **then**
    Return Clauses $[[v]$ for $v \in V]$
Initialize empty clause list C
Add clauses for at-most-n: for each combination of $n+1$ variables, add clause $[\neg v_1, \neg v_2, \ldots, \neg v_{n+1}]$
Add clauses for at-least-n: for each combination of $|V| - n+1$ negated variables, add clause $[v_1, v_2, \ldots, v_{|V|-n+1}]$
Return C

D. Experiment

We validated our approach using a 10×10 grid puzzle containing 28 numbered cells with constraint values from 0 to 7, providing coverage from highly restrictive (0 green neighbors) to demanding constraints (6-7 green neighbors).

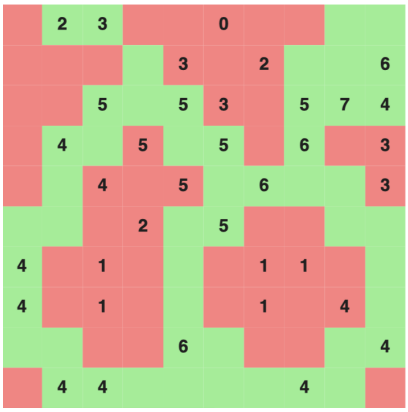

Fig. 7. Task 3 Result

The Glucose3 SAT solver successfully discovered a satisfying solution within seconds, demonstrating computational efficiency despite substantial clause complexity.

IV. Naïve Bayesian Classifier

A. Introduction and Problem Statement

This task focuses on developing a Naïve Bayesian classifier for predicting student pass (P) or fail (F) base on their score. The key challenges include discretizing continuous quiz scores, Implementing the Naïve Bayes algorithm and evaluate the model

B. Theorical Background

1) Bayes' Theorem: The Naïve Bayesian classifier is built upon Bayes' theorem:

$$P(C|X) = \frac{P(X|C) \times P(C)}{P(X)}$$

Where:

- $P(C|X)$ is the posterior probability of class C given predictor X
- $P(X|C)$ is the likelihood of predictor X given class C
- $P(C)$ is the prior probability of class C
- $P(X)$ is the prior probability of predictor X

2) Naïve Bayes Classification: For our task, We define:

- $C = \{\text{Pass}, \text{Fail}\}$ — the possible class outcomes
- $X = \{Q_1, Q_2, \ldots, Q_9\}$ — the feature vector of quiz scores

The "naïve" aspect comes from the assumption that features are conditionally independent given the class:

$$P(X|C) = P(Q_1|C) \times P(Q_2|C) \times \cdots \times P(Q_9|C)$$

This simplifying assumption allows the model to estimate the joint likelihood efficiently, even with a relatively small dataset.

C. Discretization Approach Design

To implement the Naïve Bayesian classifier for continuous quiz scores, we transform values into discrete categories.

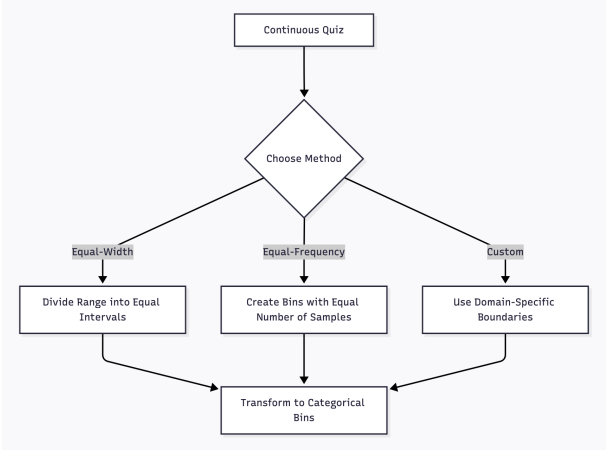

Fig. 8. Discretization flowchart

1) Discretization Methods: As illustrated in Fig. 1, our approach employs three distinct discretization strategies:

Algorithm 5 Discretize

Require: data, method, bins

for each feature in data **do**
  if method = "equal_width" **then**
    bin_edges ← linspace(min, max, bins+1) method = "equal_frequency"
    bin_edges ← percentile(feature, linspace(0, 100, bins+1))
  else
    bin_edges ← max ≤ 10 ? [0,3,5,7,10] : [0,60,70,80,90,100]
    Assign feature values to bins
    return discretized_data

The flowchart accurately represents our approach, showing how continuous quiz scores flow through one of three discretization pathways before transformed into categorical bins:

- **Equal-Width:** Divides score range in term of equal size.
- **Equal-Frequency:** Creates bins containing approximately equal numbers of samples.
- **Custom:** Uses domain-specific boundaries based on educational grading standards.

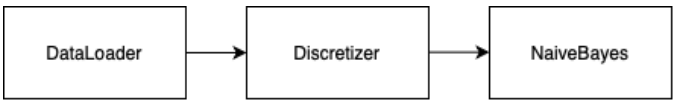

Fig. 9. Object-Oriented Design Diagram.

2) OOP Design:

- **DataLoader**: Handles dataset acquisition and preprocessing operations,
- **Discretizer**: Implements multiple discretization strategies with polymorphic behavior.
- **NaiveBayesClassifier**: Encapsulates probability computations and model evaluation metrics.

D. Experiment

1) Dataset Characteristics: The experiment utilized 267 student records (Q1-Q9), containing 177 passing students (66.3%) and 90 failing students (33.7%). Equal-width discretization with 7 bins outperformed other approaches.

2) Classification Performance: The optimal model achieved 77.90% accuracy with precision of 0.8471, recall of 0.8136, and F1-score of 0.8300. Figure 10 presents the corresponding confusion matrix.

3) Important Features: Quiz 7 (**Q7**) emerged as the most informative feature with an importance score of 0.356. In contrast, Quiz 2 (**Q2**) demonstrated the lowest predictive value (0.147).

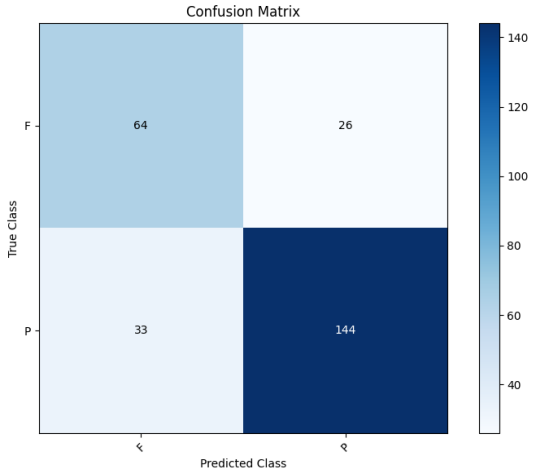

Fig. 10. Confusion matrix for the best-performing model

V. Contributions

Table I details the distribution of work and responsibilities among team members for this project.

TABLE I
Group Member Contributions

Member	Completion
Le Dang Nguyen - 522K0020	100%
Pai Hein Kyaw - 523K0078	100%
Thiha Aung - 523K0073	100%
Thin Lei Sandi - 523K0075	100%

VI. Self-evaluation

A. Task Completion Assessment

Our team has completed all project requirements: Table II show the estimated scores:

TABLE II
Self-Evaluation Scores

Component	Available	Estimated
Task 1	2.0	2.0
Task 2	3.0	3.0
Task 3	3.0	3.0
Task 4	1.0	1.0
Task 5: Report	1.0	1.0
Total	**10.0**	**10.0**

VII. Conclusion

This project successfully demonstrated the practical application of fundamental artificial intelligence techniques across diverse problem domains, revealing how different AI approaches can be applied to solve complex computational challenges.

References

[1] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671-680, May 1983.

[2] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Upper Saddle River, NJ: Pearson, 2020.