Introduction to Artificial Intelligence Final Project

Tran Thi The Nhan
Faculty of Information Technology
Ton Duc Thang University, Ho Chi Minh City, Vietnam
523k0047@student.tdtu.edu.vn

Nguyen Thi Que Chau
Faculty of Information Technology
Ton Duc Thang University, Ho Chi Minh City, Vietnam
519C0006@student.tdtu.edu.vn

Benedict Timothy Chibuike
Faculty of Information Technology
Ton Duc Thang University, Ho Chi Minh City, Vietnam
519K0078@student.tdtu.edu.vn

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

***Abstract*—This is the final project for Course Introduction to Artificial Intelligence of *Group 04*; consists of four tasks: implementing a Simulated Annealing Search on a 3D surface generated from a multivariable function using sympy and visualizing the search path; developing a 9x9 Tic-Tac-Toe game where winning requires four in a row and the computer employs a heuristic alpha-beta search with a customizable depth limit and heuristic function; solving a constraint satisfaction problem on an $m \times n$ matrix using propositional logic and the Glucose3 module of PySAT to meet neighborhood constraints; and creating a Naïve Bayesian Classifier by discretizing continuous quiz scores from a provided dataset and classifying student outcomes.**

I. Optimizing a Multivariable Function on a 3D Surface Using Simulated Annealing

A. Introduction

Task 1 of the final project aims to apply the Simulated Annealing Search (SAS) algorithm to solve a continuous optimization problem on a 2D surface. The objective is to find the coordinates (x, y) that maximize the value of the given function:

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

B. Theoretical Background: Simulated Annealing Search

Simulated Annealing (SA) is a stochastic optimization algorithm inspired by the annealing process in metallurgy, where materials are heated and slowly cooled to reach a stable, low-energy state. In optimization, this approach helps the algorithm escape local maxima by occasionally accepting worse solutions with a probability that decreases over time.

Simulated Annealing Pseudocode

function SIMULATED-ANNEALING(problem, schedule)
  return a solution state
  inputs: problem, a problem
    schedule, a mapping from time to "temperature"
  current ← MAKE-NODE(problem.INITIAL-STATE)
  for $t = 1$ to ∞ **do**
    $T \leftarrow$ schedule(t)
    if $T = 0$ **then**
      return current
    end if
    next ← a randomly selected successor of current
    $\Delta E \leftarrow$ next.VALUE − current.VALUE
    if $\Delta E > 0$ **then**
      current ← next
    else
      current ← next only with probability $e^{\Delta E/T}$

The probability of accepting a worse move is defined as:

$$e^{\Delta E/T}$$

When the temperature T is high, the algorithm explores widely and can accept many types of moves. As T decreases, the algorithm becomes more conservative, focusing the search near the best found solutions.

By shaking hard at a high temperature, the algorithm can escape local minima (red ball) and explore the search space. As the temperature decreases, the search becomes more greedy, eventually settling near the global optimum (green ball). This mechanism makes Simulated Annealing especially suitable for continuous optimization problems such as the one in Task 1, where the objective function is complex and contains multiple local optima.

This mechanism is illustrated in Fig.1 below:

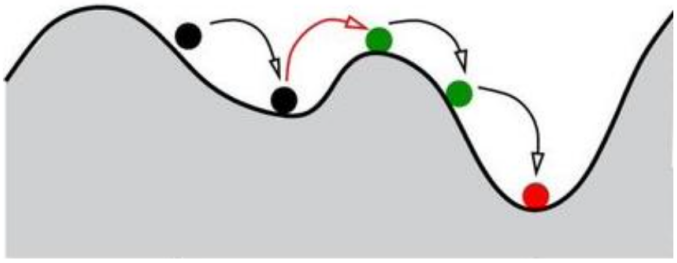


Fig. 1. Illustration of Simulated Annealing

C. Problem Definition

Task 1 aims to find the global maximum of a nonlinear function of two variables using Simulated Annealing. The search starts at the origin $(0, 0)$, explores the space with fixed steps of 0 or $\frac{\pi}{32}$, and accepts worse solutions probabilistically to escape local optima.

Implementation Overview: The solution is implemented in Python with an object-oriented design:

- `annealing.py`: Defines the `SimulatedAnnealing` class, managing search logic, temperature scheduling, and tracking best states.
- `main.py`: Coordinates execution.
- `surface_plot.py`: Visualizes the 3D surface, search path, and optimal point.

Visualization: A 3D plot displays the function surface, the search path, start point (blue), and the best point found (dark red).

D. Technical Implementation

Our implementation uses object-oriented design with three core modules:

1) **SimulatedAnnealing Class:** Core logic of the algorithm.
2) **Surface Plotting:** Uses Plotly for interactive visualization.
3) **Main Program:** Defines the objective function and controls execution.

The SimulatedAnnealing class is the core component, encapsulating all the logic for the optimization algorithm. It maintains the current state, best state found, and search path while providing methods for the annealing process. The plotting function is separated into its own module for better code organization, and the main program orchestrates the overall execution flow.

E. Experimental Results

After running the Simulated Annealing algorithm, the following results were obtained:

- **Best Coordinates:** $(x^*, y^*) = (-2.0617, 3.4361)$
- **Maximum Function Value:** $f(x^*, y^*) = 2.7872$
- **Steps Taken:** 113 iterations
- **Visualization:** The figure below shows the 3D surface of the objective function, with the red line representing the search path, the blue dot as the starting point, and the dark red dot as the best point found.

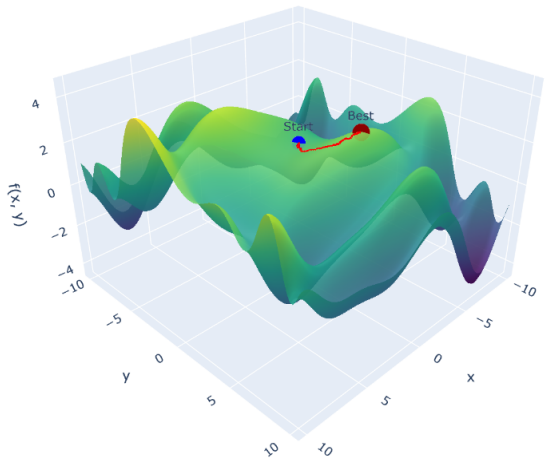


Fig. 2. 3D Surface of the objective function

Analysis

The results demonstrate that Simulated Annealing effectively escapes local maxima and converges to a near-global optimum. The search path indicates exploration across the function landscape before settling near the best solution.

II. Real-Time Optimized AI for Large-Scale Tic Tac Toe Using Heuristic Alpha-Beta Search

A. Objective and Challenges

The goal is to ***develop a strategic AI for 9×9 Tic Tac Toe (win condition: 4 in a row) that responds within 3 seconds***. Key challenges include handling a vast search space (81 initial moves), ensuring real-time performance, maintaining strategic depth for multi-step threat detection, and isolating simulation from actual game state. The AI must be both efficient and competitive against human players.

B. Solution Approach

Firstly, the word ***'pruning'*** means cutting down branches and leaves. In Artificial Intelligence, Alpha-beta pruning is the pruning of useless branches in decision trees. This alpha-beta pruning algorithm was discovered independently by researchers in the 1900s.

Alpha Beta Pruning is a search optimization technique that improves the performance of the minimax algorithm. The minimax algorithm is a decision-making process commonly used in two-player, zero-sum games like chess. In such games, one player aims to maximize their score while the other seeks to minimize it.

We have tested the game on a computer has Windows operating systems has *13th Gen Intel(R) Core(TM) i7-13620H*

Fig. 3. An illustration of alpha–beta pruning. The grayed-out subtrees don't need to be explored (when moves are evaluated from left to right), since it is known that the group of subtrees as a whole yields the value of an equivalent subtree or worse, and as such cannot influence the final result. The max and min levels represent the turn of the player and the adversary, respectively. *Source: Wikipedia*

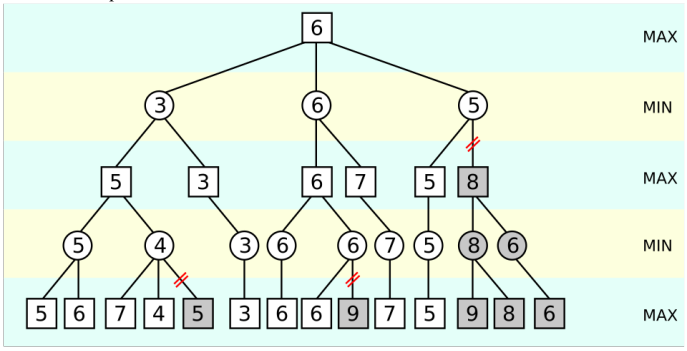
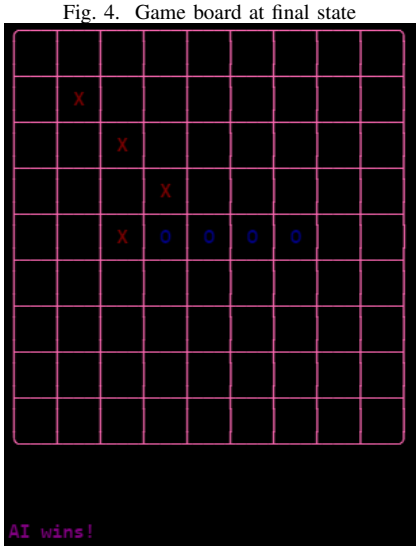


Fig. 4. Game board at final state



CPU. The game operates on a control screen. The player selects a square by entering the coordinates of that square as the row and column index numbers from the keyboard.

Solution steps

1) Define a `TicTacToe` class to encapsulate board state, players, and game logic.
2) Initialize a 9×9 NumPy array for the board and precompute all length-4 winning lines.
3) Implement move validation and board-drawing routines (e.g., using the `rich` library).
4) Create a heuristic evaluation function that scores each length-4 line based on token counts, empties, and blocked lines.
5) Implement Alpha-Beta pruning with a recursive `alpha_beta(board, depth,` α`,` β`, maximizing)` function.
6) Wrap the Alpha-Beta search in iterative deepening to enforce the 3s time limit.
7) Ensure state isolation by copying the board (`np.copy`) before simulating each move.
8) Order legal moves by their distance to the center to enhance pruning effectiveness.
9) After each move, check for a win (4 in a row) or a full board (draw).
10) Loop between user-input moves and AI move selection until the game concludes.

After 8 moves of us and the computer, **the computer won** as in **Figure 4** below. Average computer move time was 3.0 seconds, highest time was 3.24 seconds, fastest was 3.04 seconds.

III. Constraint Satisfaction Problems with Propositional Logic

A. Introduction

This task involves coloring cells in an $m \times n$ matrix, where each cell contains a non-negative integer or is blank. The goal is to color cells red or green so that each numbered cell has exactly that many green cells in its 3x3 neighborhood. Blank cells impose no constraints.

We model this as a propositional logic problem and solve it using the Glucose3 SAT solver from PySAT. Each cell is a propositional variable, and constraints for numbered cells are encoded in Conjunctive Normal Form (CNF) for the solver.

B. Theoretical Background

A **Constraint Satisfaction Problem (CSP)** consists of variables with defined domains and constraints that must be satisfied. In this task, each cell is a Boolean variable (green/red), and the constraint requires numbered cells to have a specific number of green neighbors.

The CSP is converted into propositional logic, with constraints expressed in CNF and solved using a SAT solver like Glucose3.

C. Problem Definition

The matrix coloring task can be formulated as a CSP with the following specifications:

- **Variables**: Each cell in the $m \times n$ matrix is treated as a Boolean variable, representing whether the cell is green (True) or red (False).
- **Domain**: True, False (green, red).
- **Constraints**:
  - For each cell containing a non-negative integer n, the sum of green cells among itself and its eight immediate neighbors must be exactly n.
  - Blank cells impose no constraints.
- **Input**:
  - An $m \times n$ matrix, where each cell contains either a non-negative integer or is left blank.
- **Output**:
  - A coloring of the matrix (each cell assigned green or red) that satisfies all constraints, or a report if no solution exists.

To model the constraints, for each numbered cell, we generate logical clauses to ensure that exactly n of its nine neighbors (including itself) are assigned True (green). All generated clauses are encoded in CNF for processing by the SAT solver.

D. Technical Implementation

The solution follows an object-oriented design with the following components:

- **main.py**: Loads the input matrix, initializes the solver, and manages execution.
- **CSP_Solver class**:
  - Encodes constraints to CNF from the input matrix.
  - Uses PySAT's Glucose3 solver to find valid colorings.
  - Outputs the final solution.
- **Utilities**: Handle matrix input, neighbor detection, and console visualization.

Workflow Summary:

1) Load matrix and assign Boolean variables.
2) Encode neighbor constraints for numbered cells.
3) Add CNF clauses to the SAT solver and run it.
4) Display the resulting color assignment or report failure.

The modular design ensures clarity, reusability, and easy adaptation to similar CSP problems.

E. Experimental Results

After running the SAT-based CSP solver on a sample input matrix, we obtained the following results:

Sample Input Matrix: (A dot "." denotes a blank cell, numbers are constraints)

```
.  2  3  .  .  0  .  .  .  .
.  .  .  .  3  .  2  .  .  6
.  .  5  .  5  3  .  5  7  4
.  4  .  5  .  5  .  6  .  3
.  .  4  .  5  .  6  .  .  3
.  .  .  2  .  5  .  .  .  .
4  .  1  .  .  .  1  1  .  .
4  .  1  .  .  .  1  .  4  .
.  .  .  .  6  .  .  .  .  4
.  4  4  .  .  .  .  4  .  .
```

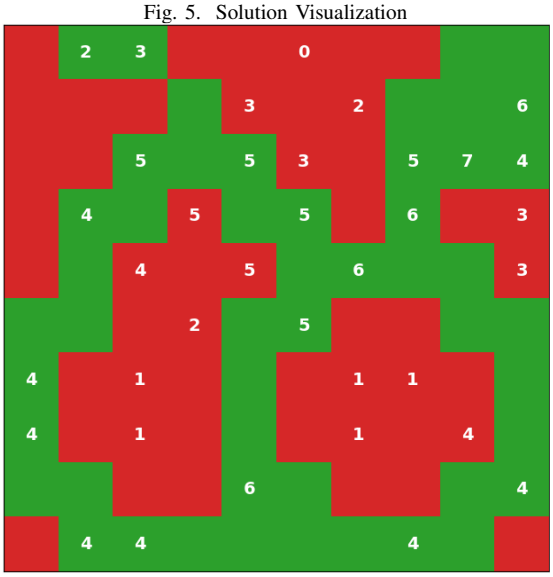
Solution:
Number of clauses generated: 3684
Total number of variables: 100

IV. Object-Oriented Implementation of Naïve Bayes for Predicting Course Results from Quiz Data

A. Problem Definition

This task involves predicting student performance outcomes—Pass (P) or Fail (F)—based on their scores across nine quizzes (Q1 to Q9), given in the `task4_data.csv` dataset. All quiz scores are continuous values, making direct application of Naïve Bayes (which assumes categorical features) unsuitable without preprocessing.

Fig. 5. Solution Visualization



The Naïve Bayes classifier is a probabilistic machine learning algorithm based on Bayes' Theorem, assuming independence among features given the class label. The algorithm calculates the posterior probability for each class based on the observed feature values and chooses the class with the highest probability. The core formula is:

$$P(C \mid X) = \frac{P(C) \cdot P(X \mid C)}{P(X)}$$

where $P(C \mid X)$ is the probability of class C given features X, $P(C)$ is the prior probability of the class, $P(X \mid C)$ is the likelihood of the features given the class, and $P(X)$ is the marginal probability of the features. In practice, $P(X)$ is constant across classes and thus often omitted from the calculation.

The ***"naïve"*** assumption that all features are conditionally independent simplifies the computation of the likelihood, making the algorithm fast and scalable.

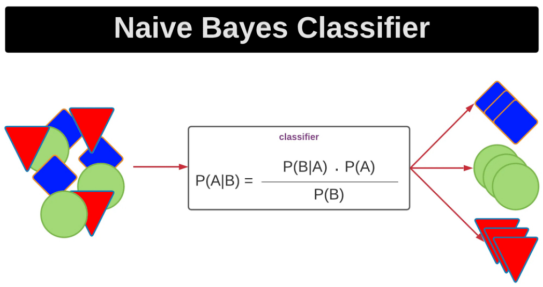


Fig. 6. What is Naive Bayes algorithm?
Source:https://mlarchive.com/machine-learning/the-ultimate-guide-to-naive-bayes/

B. Proposed Approach

To address the classification challenge, a quantile-based discretization method is employed to convert each quiz score into discrete bins. This transformation enables the application of a Naïve Bayes classifier.

An object-oriented Naïve Bayes model is implemented, calculating class priors and conditional feature probabilities using **Laplace smoothing.**

The prediction process selects the class with the highest log posterior probability, offering a numerically stable computation method.

C. Evaluation

The dataset is split into training and test sets using a stratified 70/30 ratio. The classifier yields the following performance:

- **Accuracy:** 80.25%
- **Precision (P/F):** 0.82 / 0.76
- **Recall (P/F):** 0.91 / 0.59
- **F1-Score (P/F):** 0.86 / 0.67

D. Visualization and Feature Analysis

To understand the dataset, we visualized some plots, one of them are the class distribution and the distribution of each quiz score by class. The class distribution plot shows a moderate imbalance, with more students passing than failing.

A feature importance analysis was conducted based on the differences in conditional probabilities across classes. As illustrated in Figure 7 below, quiz scores such as Q9 and Q7 exhibit higher discriminative power between the `Pass` and `Fail` classes. This insight helps identify which assessments most influence classification outcomes.

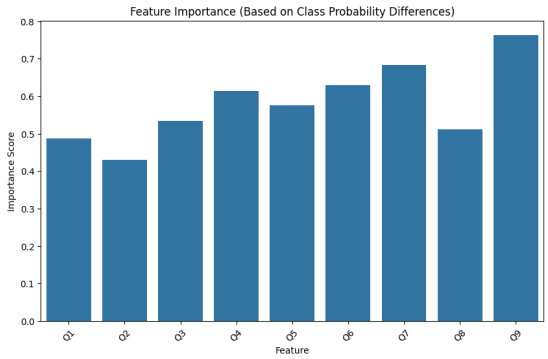


Fig. 7. Feature Importance Based on Class Probability Differences

V. Contributions

Group 4 Contributions		
Member Name	**Tasks**	**Completion Levels**
Tran Thi The Nhan	Task 2, Task 5	100%
Nguyen Thi Que Chau	Task 1, Task 3	100%
Benedict Timothy Chibuike	Task 4	80%

VI. Self-evaluation

Based on the distribution of tasks and the actual contributions made by each group member, we have completed all assigned responsibilities at a high level of quality. Our team has completed all project requirements:

- **Task 1 (2.0 point(s))**: Implemented Simulated Annealing Search on 3D surfaces
- **Task 2 (3.0 point(s))**: Implemented a program that allows users to play Tic-Tac-Toe against the computer on a 9 × 9 board with Heuristic Alpha-Beta Search
- **Task 3 (3.0 point(s))**: Solved Constraint Satisfaction Problems with Propositional Logic
- **Task 4 (1.0 point(s))**: Implemented and compute accuracy of Naïve Bayesian Classifier
- **Task 5 (1.0 point(s))**: Composed the project report using the IEEE conference proceeding template

Estimated Group Score: 100%

VII. Conclusion

This project successfully implemented four core AI techniques: Simulated Annealing for 3D function optimization, Alpha-Beta Search for strategic game AI, Propositional Logic for CSP solving, and Naïve Bayes for educational outcome prediction. Key achievements include:

- 100% task completion with all objectives met
- Demonstrated effectiveness through quantitative results (80.25% classification accuracy, 3-second move times)
- Maintained theoretical rigor while delivering practical implementations

References

[1] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Phil. Trans. Roy. Soc. London*, vol. A, 1975. [Online]. Available: https://kodu.ut.ee/~ahto/eio/2011.07.11/ab.pdf

[2] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959. [Online]. Available: https://www.researchgate.net/publication/224103556_Arthur_Samuel_Pioneer_in_Machine_Learning

[3] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2020. [Lecture slides]. Available: https://people.engr.tamu.edu/guni/csce625/slides/AI.pdf

[4] By Great Learning Editorial Team, *Alpha Beta Pruning in AI*, January 23, 2025. https://www.mygreatlearning.com/blog/alpha-beta-pruning-in-ai/

[5] "What is Simulated Annealing?", studocu.vn. Ho Chi Minh City, March 3, 2023 https://www.studocu.vn/vn/document/truong-dai-hoc-su-pham-ky-thuat-thanh-pho-ho-chi-minh/nhap-mon-cntt/simulated-annealing-huhu/56828778

[6] Lecoutre, Christophe (2013). Constraint Networks: Techniques and Algorithms. Wiley. p. 26. https://cse.unl.edu/~choueiry/Documents/Lecoutre-eBook-Constraint%20Networks.pdf

[7] Marcin J. Mizianty, Lukasz A. Kurgan and Marek R. Ogiela. Discretization as the enabling technique for the Naïve Bayes and semi-Naïve Bayes-based classification. Published online by Cambridge University Press: 01 December 2010. https://www.cambridge.org/core/journals/knowledge-engineering-review/article/abs/discretization-as-the-enabling-technique-for-the-naive-bayes-and-seminaive-bayesbase82DD7C5629E3331A50E7CE170E84B11D?utm_campaign=shareaholic&utm_medium=copy_link&utm_source=bookmark

[8] Zhang, Harry. The Optimality of Naive Bayes (PDF). FLAIRS2004 conference. https://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf