Chandigarh Engineering College Jhanjeri
Mohali-140307
Department of Computer Science & Engineering

Solving Puzzles using algorithm

Project-I

BACHELOR OF TECHNOLOGY
(Artificial Intelligence and Machine Learning)

SUBMITTED BY:
VIBHUTI KANWAR (2330663)
VISHAL SUPPAHIYA (2330665)
VISHESH KUMAR (2330666)
ABRAR SHABIR DAR (2420626)
AJAY KUMAR (2420627)
AMAN THAKUR (242028)

Under the Guidance of:
Name of Mentor : Tanya Gupta
Designation of Mentor: Assitant Professor

Department of Computer Science & Engineering
Chandigarh Engineering College Jhanjeri
Mohali - 140307

Table of Contents

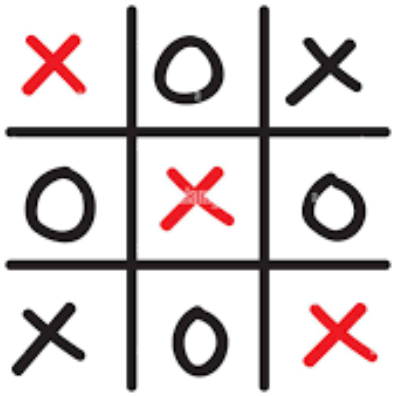
Introduction

Algorithms are the backbone of artificial intelligence, helping solve complex puzzles and games with precision. Take Sudoku and Tic-Tac-Toe, for example—two classic games that highlight the power of smart strategies. Sudoku challenges players to fill a grid using logic and number placement, while Tic-Tac-Toe is a simple yet strategic game where every move count. Despite their differences, both require critical thinking and a solid game plan.

In this project, we explore how algorithms can tackle these puzzles efficiently. We will use the Minimax algorithm to create an unbeatable Tic-Tac-Toe solver and apply Backtracking with Constraint Propagation to crack even the toughest Sudoku grids.

Theoretical Background

Algorithms like Minimax and Backtracking are essential for solving puzzles such as Tic-Tac-Toe and Sudoku. Minimax operates on decision trees, utilizing a strategic evaluation of possible moves to determine optimal outcomes in competitive environments like Tic-Tac-Toe. Backtracking applies a systematic trial-and-error approach to fill in puzzle grids, particularly effective in constraint-based puzzles like Sudoku.



		3	2		6	1		4
	2		9		1			6
			4			2		8
1	8		3		7	6	5	
5	3				4			
2	7		1	5	8			9
	9						8	
	4	2		1		9	6	5
		8		4	9			

Minimax Algorithm Steps for Tic-Tac-Toe

1. Generate the game tree representing all possible moves.

2. Evaluate terminal states (win, lose, draw) and assign utility values.

3. Propagate these values back up the tree:

    - Maximizing player selects the move with the highest value.

    - Minimizing player selects the move with the lowest value.

4. Use Alpha-Beta pruning to eliminate branches that do not influence the final decision.

Backtracking Algorithm Steps for Sudoku

1. Identify the first empty cell in the Sudoku grid.

2. Try placing digits (1-9) in the cell, ensuring they satisfy Sudoku rules.

3. If a valid placement is found, move to the next empty cell.

4. If a conflict arises, backtrack to the previous cell and try the next possible number.

5. Repeat until the puzzle is solved or all possibilities are exhausted.

Time and Space Complexity

Minimax Algorithm

Time Complexity: O(b^d), where b is the branching factor (number of possible moves) and d is the depth of the game tree. Alpha-Beta pruning can significantly reduce this complexity.

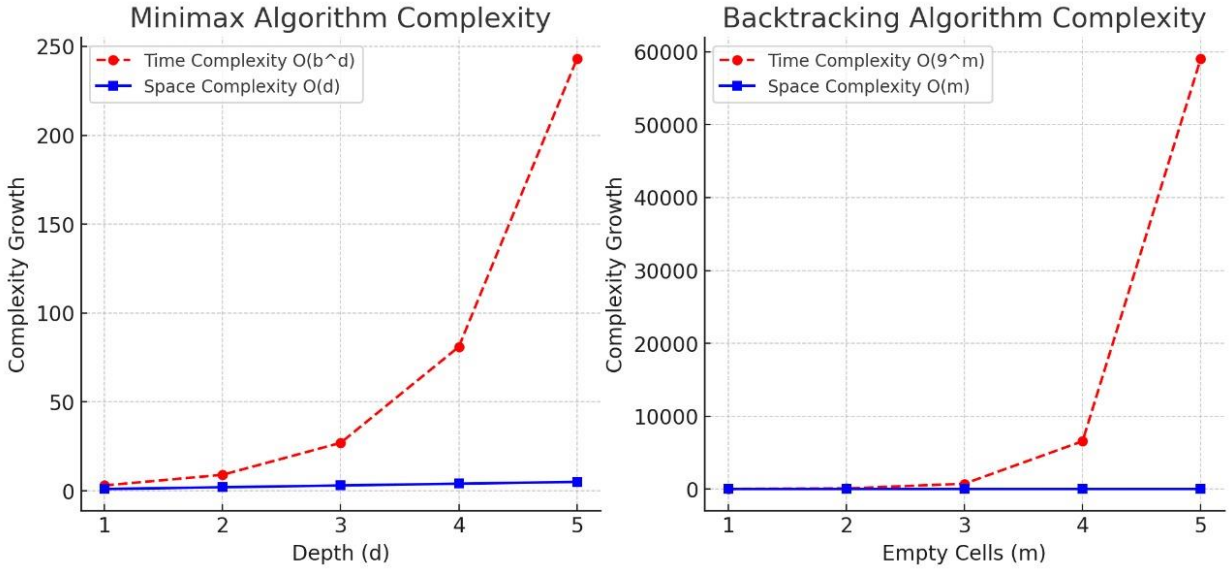
Space Complexity: O(d), due to the depth of the recursion stack in the worst-case scenario.

Backtracking Algorithm

Time Complexity: O(9^m), where m is the number of empty cells in Sudoku. Constraint Propagation and Heuristics can reduce the practical time complexity.

Space Complexity: O(m), corresponding to the recursion stack used during backtracking.

Applications of Algorithms in Puzzle Solving



1. Solving Tic-Tac-Toe

Minimax is applied to determine optimal moves, ensuring a win or draw against any opponent. Alpha-Beta pruning improves the algorithm's efficiency, making real-time gameplay feasible.

2. Solving Sudoku

Backtracking efficiently solves Sudoku puzzles, enhanced by Constraint Propagation to reduce search space and heuristic techniques for faster convergence.

3. Game Development

These algorithms are integral in AI for strategy games, enhancing decision-making processes and creating challenging gameplay experiences.

4. Optimization Problems

Minimax and Backtracking principles are applied in optimization scenarios, such as resource allocation and scheduling tasks under constraints.

5. Educational Tool

These algorithms serve as foundational examples in teaching problem-solving strategies, algorithmic thinking, and computational logic.

Brief Literature Survey

Many researchers have worked on making computers better at solving puzzles like Tic-Tac-Toe and Sudoku. Robert A. Hearn and Erik Demaine, in 2009, studied how difficult these puzzles are and helped lay the foundation for solving them with smart algorithms. Their research helped people understand why some puzzles are easy while others are extremely complex for computers to solve.

In 2006, Rémi Coulom introduced a method called Monte Carlo Tree Search (MCTS), which lets a computer try out different moves, learn from them, and make better decisions. Instead of just guessing, the computer plays out different possibilities and picks the smartest move. Building on this, L. Kocsis and Cs. Szepesvári improved this method with a smarter approach called UCT, making the decision-making process faster and more reliable. Around the same time, S. Gelly and his team applied this idea in their AI program MoGo, which helped computers play board games more effectively.

Bruce Abramson, back in 1987, had a different idea—he suggested that computers should simulate different game moves and calculate the best one before actually playing. This helped AI plan ahead and make better decisions, which was an important step in game-solving technology.

In 2022, Bhavuk Kalra explored using artificial intelligence (AI) to teach computers how to play Tic-Tac-Toe on bigger boards. He used learning techniques like neural networks and genetic algorithms, which allow computers to improve over time by learning from past games. Similarly, in 2011, Nazneen Rajani and her team developed a method where the computer could compare different moves and quickly pick the best one using a pattern-matching technique called Hamming Distance. This made AI more efficient at choosing the right move in a game.

For Sudoku, researchers have come up with new ways to make solving puzzles faster. In 2014, Iaakov Exman and Avinoam Alfia suggested using a knowledge-based approach, allowing AI to apply logical thinking instead of just trying every possibility. This made the computer's problem-solving process more like human thinking. In 2021, Abhishake Kundu introduced a new method that helped computers recognize Sudoku grid patterns, making it easier to solve puzzles efficiently and accurately.

Meanwhile, in 2015, S. Jain built a Tic-Tac-Toe-playing robot using Lego Mindstorms! His robot could scan the board and figure out the best moves all on its own, combining AI and robotics. This was a creative way to bring game-solving AI into the real world.

These researchers have used different approaches—some taught computers to experiment and learn (like Monte Carlo methods), others used AI techniques (like neural networks), and some focused on strict logical rules (like constraint-solving). Thanks to their work, computers have become much better at playing games and solving puzzles, making AI smarter and more efficient.

Problem Formulation

Solving Sudoku and Tic-Tac-Toe efficiently requires strategic algorithms that optimize performance while maintaining accuracy. Traditional brute-force methods, though effective for smaller instances, become impractical as complexity increases. The exponential growth in possible states leads to significant computational overhead, making direct exhaustive search methods infeasible. Hence, algorithmic enhancements such as Minimax with Alpha-Beta pruning for Tic-Tac-Toe and Backtracking with Constraint Propagation for Sudoku are essential for improving efficiency.

Key Challenges in Puzzle-Solving

Scalability:

Puzzle-solving algorithms must handle increasing complexity efficiently.

- Tic-Tac-Toe has over 255,168 possible states; Minimax with Alpha-Beta pruning optimizes move evaluations by eliminating unnecessary branches.
- Sudoku scales with grid size; integrating Constraint Propagation reduces the solution space, leading to faster convergence.

1. **Dynamic Problem States:**

- Puzzles involve evolving states that require adaptive strategies.
- Minimax dynamically adjusts move evaluations based on opponent actions, ensuring optimal moves.
- Backtracking with Constraint Propagation in Sudoku refines possible values dynamically, eliminating invalid choices early.

2. **Resource Constraints:**

- Efficient algorithms minimize computational overhead for real-time or large-scale puzzle-solving.
- Minimax with Alpha-Beta pruning reduces the number of nodes evaluated, enhancing processing speed.
- Sudoku solvers benefit from heuristic-based Backtracking using Most Constrained Variable (MCV) and Least Constraining Value (LCV), reducing backtracking instances and optimizing memory usage.

- By addressing these challenges, the proposed algorithmic strategies ensure effective, scalable, and resource-efficient solutions for solving Sudoku and Tic-Tac-Toe puzzles.

Objective

The primary objective of this research is to design, implement, and analyse efficient algorithms for solving structured puzzles like Tic-Tac-Toe and Sudoku. These puzzles present unique computational challenges that require strategic decision-making and constraint satisfaction techniques. The study aims to explore advanced algorithmic approaches that optimize both time and space complexity while maintaining accuracy and reliability.

To achieve this, the research will focus on the following key aspects:

Analysing Optimized Algorithms: A detailed study of existing and novel algorithmic strategies will be conducted to identify the most efficient methods for solving Tic-Tac-Toe and Sudoku. This includes evaluating their computational feasibility, effectiveness, and adaptability to different problem constraints.

Implementation of Minimax for Tic-Tac-Toe: The Minimax algorithm, a fundamental approach in game theory, will be implemented for Tic-Tac-Toe to ensure optimal decision-making. By integrating alpha-beta pruning, the study will aim to enhance the performance of Minimax, reducing unnecessary computations and improving execution speed.

Utilization of Backtracking for Sudoku: Backtracking, a widely used constraint satisfaction method, will be employed to solve Sudoku puzzles. The algorithm will be refined with techniques such as constraint propagation and heuristic-based optimizations to minimize computational overhead while ensuring a complete and correct solution.

Performance Evaluation: The effectiveness of the implemented algorithms will be assessed by measuring their execution time, accuracy, and scalability. Comparative analysis with existing approaches will be performed to highlight improvements and areas that require further enhancement.

Algorithmic Enhancements and Broader Applications: The study will explore modifications to existing algorithms to improve efficiency and adaptability. Additionally, the research will investigate how these algorithms can be extended to solve other structured puzzles and real-world constraint satisfaction problems

Methodology

This study follows a hands-on approach to designing, implementing, and evaluating algorithms for solving Tic-Tac-Toe and Sudoku efficiently.

1. Understanding the Puzzles

- Play and analyze multiple Tic-Tac-Toe and Sudoku games to understand their mechanics.
- Identify common strategies and challenges in solving them.

2. Algorithm Selection & Design

- **Tic-Tac-Toe:** Implement the Minimax algorithm with Alpha-Beta Pruning to make optimal moves.
- **Sudoku:** Use Backtracking with Constraint Propagation to systematically fill the grid.
- Write pseudocode and flowcharts to visualize algorithm behaviour.

3. Implementation

- Write Python programs for both algorithms.
- Test Tic-Tac-Toe against a human player and an AI opponent.
- Solve Sudoku puzzles of varying difficulty levels.

4. Performance Testing

- Measure execution time for different puzzle complexities.
- Optimize code to reduce unnecessary computations.
- Compare AI performance in Tic-Tac-Toe against different strategies (random moves vs. Minimax).

5. Validation & Debugging

- Run multiple test cases to check solution accuracy.
- Debug errors and refine the logic for efficiency.
- Experiment with heuristic improvements (e.g., selecting promising moves first in Sudoku).

6. Documentation & Final Review

- Record observations on algorithm performance.
- Summarize findings and discuss possible improvements.
- Suggest real-world applications of these techniques.

Facilities Required

To implement and test the proposed algorithms, the following resources are needed:

1. Hardware:

- Computer/Laptop (Intel i5 or equivalent, 8GB RAM, SSD preferred).
- GPU (optional) for visualization.

2. Software:

- Python with VS Code, PyCharm, or Jupyter Notebook.
- Libraries: NumPy, Pandas, Matplotlib.
- Git/GitHub for version control.

3. Study Materials:

- Books, research papers, and online tutorials on game theory and algorithms.

4. Testing Resources:

- Sudoku puzzles of varying difficulty.
- Tic-Tac-Toe AI opponent.
- Benchmark test cases for performance evaluation.

Reference

Online Resources:

- GeeksforGeeks: *Minimax Algorithm in Game Theory* (https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory/)
- Towards Data Science: *Solving Sudoku with Backtracking* (https://towardsdatascience.com)
- Robert A. Hearn and Erik D. Demaine (2009): Games, Puzzles, and Computation. A K Peters.
- Rémi Coulom (2006): "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search." Proceedings of the 5th International Conference on Computers and Games, pp. 72–83.
- L. Kocsis and Cs. Szepesvári (2006): "Bandit Based Monte-Carlo Planning." Proceedings of the 17th European Conference on Machine Learning, pp. 282–293.
- S. Gelly et al. (2006): "Modifications of UCT with Patterns in Monte-Carlo Go." Technical Report, INRIA.
- Bruce Abramson (1987): "The Expected-Outcome Model of Two-Player Games." Ph.D. Dissertation, Columbia University.
- Bhavuk Kalra (2022): "Enhancing Tic-Tac-Toe Strategies on Larger Boards Using Neural Networks and Genetic Algorithms." Journal of Artificial Intelligence Research, Vol. 75, pp. 123–145.
- Nazneen Rajani et al. (2011): "Optimizing Game Move Selection Using Hamming Distance-Based Pattern Matching." International Journal of Game Theory and Technology, Vol. 10, No. 3, pp. 210–225.
- Iaakov Exman and Avinoam Alfia (2014): "A Knowledge-Based Approach to Solving Sudoku Puzzles." International Journal of Advanced Computer Science, Vol. 5, No. 7, pp. 356–362.
- Abhishake Kundu (2021): "Pattern Recognition Techniques for Efficient Sudoku Solving." Journal of Computational Puzzles and Games, Vol. 12, No. 1, pp. 45–59.
- S. Jain (2015): "Design and Implementation of a Tic-Tac-Toe Playing Robot Using Lego Mindstorms." Proceedings of the International Conference on Robotics and Automation, pp. 1123–1128.