Faculty of Engineering & Technology

Electrical & Computer Engineering Department

ENCS3340 - Artificial intelligence

Project Report

Prepared by :

Hanan Alawawda - 1230827

Amjad Adi – 1230800

Instructor: Aziz Qaroush

Section: 2

Date : 12-12-2005

Table of Contents

1. **Formulation of the Problem..1**

1.1 CSP Formulation..1

1.1.2 CSP Backtracking and constraint violations...1

1.1.3 CSP Heuristics..2

1.2 Simulated Annealing..3

1.2.1 SA Formulation..3

1.2.2 SA Cost Function and constraint violations..3

1.2.3 SA Neighbor generation and acceptance rule...3

1.2.4 SA Parameters and stopping criteria..4

2. **Experimental Results and Graphs..4**
3. **Comparative Discussion...9**
4. **Conclusion..10**

Table of Figures

Experimental Results and Graphs

Figure 1: Experimental Results and Graphs - Time Comparison(9X9)..................................4

Figure 2: Experimental Results and Graphs - Completeness Comparison (9X9)..........................5

Figure 3: Experimental Results and Graphs - Constraints Violated Comparison (9X9)................5

Figure 4: Experimental Results and Graphs - Iterations/Backtracks Comparison (9X9)................6

Figure 5: Time Comparison(16X16) 6

Figure 6: Completeness Comparison (16X16)...7

Figure 7: Constraints Violated Comparison (16X16)...7

Figure 8: Iterations/Backtracks Comparison (16X16)..8

1. **Formulation of the Problem**

Constraint Satisfaction Problem (CSP) Backtracking and Simulated Annealing (SA) are two AI techniques used to solve a 9x9 Sudoku puzzle. The objective is to satisfy the Sudoku constraints by filling a 9x9 grid so that each row, column, and 3x3 sub grid contains every digit from 1 to 9 exactly once.

1.1 CSP

1.1.1 CSP Formulation

The Sudoku problem is formulated as a list of lists (rows of cells) where each grid cell is a variable. The variable is assigned a value from 1 to 9 or a '.' if no assignment has been made to it. Each unassigned variable domain is the range of potential values from 1 to 9.

1.1.2 CSP Backtracking and constraint violations

In constraint violations, the algorithm proceeds implicitly: it only continues and assigns a value if that value does not violate any constraints. If a violation occurs, it backtracks to restore consistency, the following constrains are.

- Row constraint: Every number appears in every row exactly once.
- Column constraint: Every number appears in every column exactly once.
- Box constraint: Each 3×3 subgrid contains exactly one instance of each number.

1.1.3 CSP Heuristics

This method uses three heuristics to enhance the search and Backtracking Search to assign values to the variables:

Minimum Remaining Values (MRV): The variable that has the fewest legal values left is chosen next. The goal of this heuristic is to steer clear of decisions that could result in dead ends.

Most Constraining Variable: The variable that holds the most constraints on his unassigned neighbors is selected. because choosing it early tends to reduce the branching factor and expose dead-ends sooner.

Least Constraining Value (LCV): The value that leaves the greatest number of options for other variables is selected when choosing a value for a variable.

Forward checking: After assigning a value to a cell, we immediately remove that value from the domains of all unassigned neighbors. If any neighbor's domain becomes empty, we stop that path and backtrack immediately, restoring domains when undoing the assignment.

In this project, MRV is applied first to select the next variable; when multiple variables tie, the tie is resolved using MCV. Afterward, the value is selected using LCV, and forward checking is performed after each assignment to prune inconsistent domains early and detect dead-ends as soon as possible.

The combination of MRV, MCV, LCV, and forward checking significantly improves performance by shrinking the search space and reducing wasted exploration. MRV lowers the branching factor by prioritizing the most constrained cells, MCV helps break ties by selecting the cell that most influences the remaining unassigned neighbors, and LCV preserves flexibility by choosing values that eliminate the fewest options for others. Meanwhile, forward checking prunes inconsistent domain values immediately after each assignment, allowing dead-ends to be detected early rather than deep in the recursion, which will be seen in the results part.

To evaluate the impact of CSP heuristics, an additional experiment was conducted using pure backtracking without heuristics. A maximum limit of 300,000 backtracking steps was enforced to reduce runtime; if this limit was reached, the algorithm terminated and returned failure.

1.2 Simulated Annealing

1.2.1 SA Formulation

The Sudoku problem is formulated as a list of lists (rows of cells), where each cell represents a variable. Unlike CSP, Simulated Annealing treats Sudoku as an optimization problem by working on a complete board configuration and aiming to reduce energy/cost function. The cost function measures how far the current state is from a valid Sudoku solution by counting constraint violations and penalizing unassigned cells.

1.2.2 SA Cost Function and constraint violations

Constraint violations are measured using a cost function that counts duplicates and inconsistencies across the Sudoku constraints. In this formulation, violations are calculated by counting repeated values in each row, column, and 3×3 box, where duplicates increase the total cost. In addition, every unassigned cell (represented by '.') is counted as an extra penalty, which encourages the algorithm to move toward complete assignments. A solution is considered valid when the board contains no empty cells and the total number of violations becomes zero, meaning all constraints are satisfied.

Row constraint: Every number appears in every row exactly once.
Column constraint: Every number appears in every column exactly once.
Box constraint: Each 3×3 subgrid contains exactly one instance of each number.

1.2.3 SA Neighbor generation and acceptance rule

The search proceeds by repeatedly generating a neighbor state using small random modifications to the current board. A neighbor is created by selecting a non-fixed position and either assigning a value that is consistent with local constraints (when possible), or selecting a random value when no legal option is available. Each neighbor is evaluated immediately using the cost function. If the new state reduces the cost, it is accepted directly. If the cost increases, the move may still be accepted with a probability that depends on the current temperature, which allows escaping local minima early in the search. This probability decreases gradually as the temperature cools down, making the algorithm more selective as it progresses.

1.2.4 SA Parameters and stopping criteria

The SA behavior is controlled mainly by three parameters: initial temperature, cooling rate, and maximum iterations. A higher initial temperature increases the chance of accepting worse moves at the beginning, while the cooling rate controls how quickly this randomness is reduced during the run. The algorithm terminates when a valid solution is found (cost = 0), when the temperature becomes extremely small, or when the maximum number of iterations is reached.

2. Experimental Results and Graphs

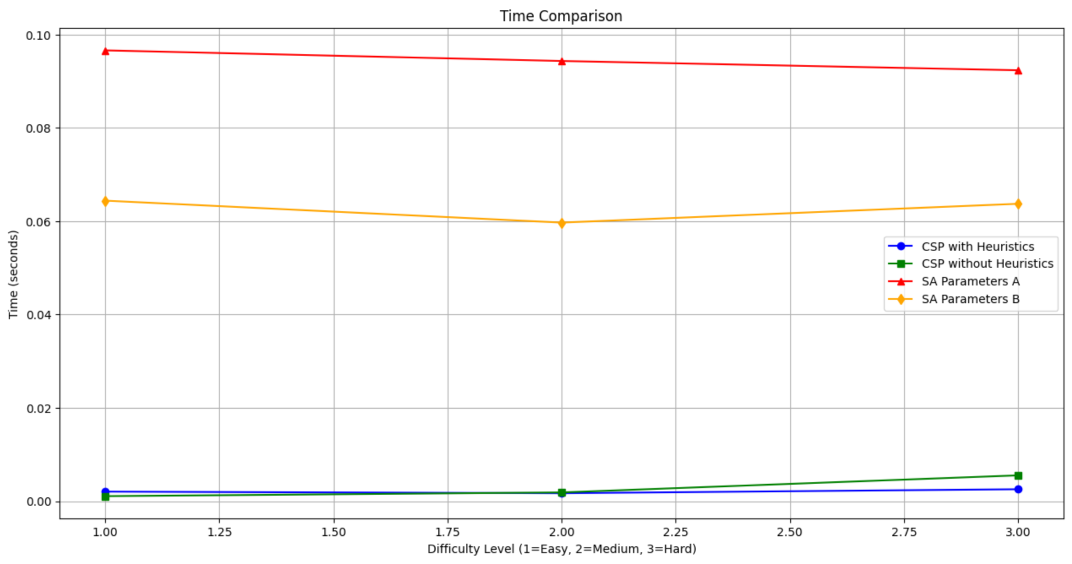

Figure 1 : Experimental Results and Graphs - Time Comparison (9X9)

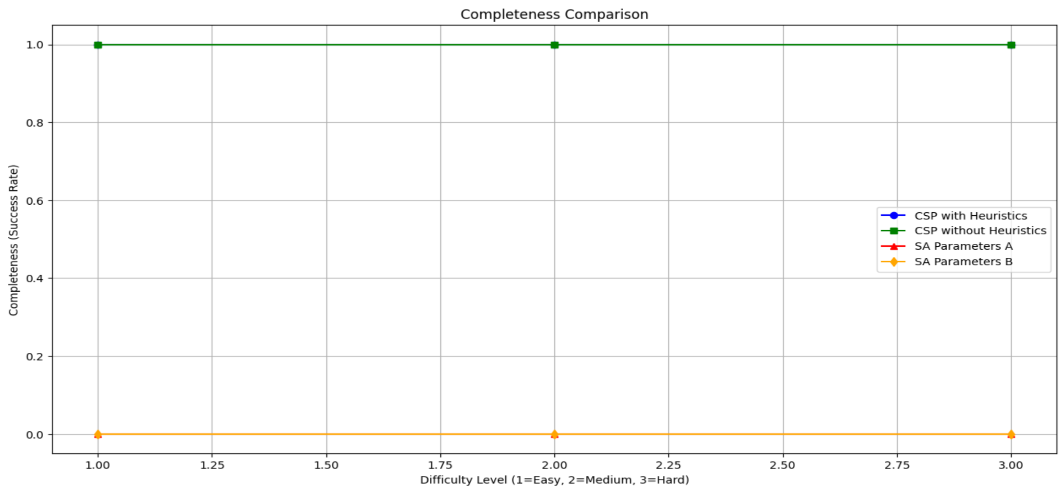

Figure 2: Experimental Results and Graphs - Completeness Comparison (9X9)

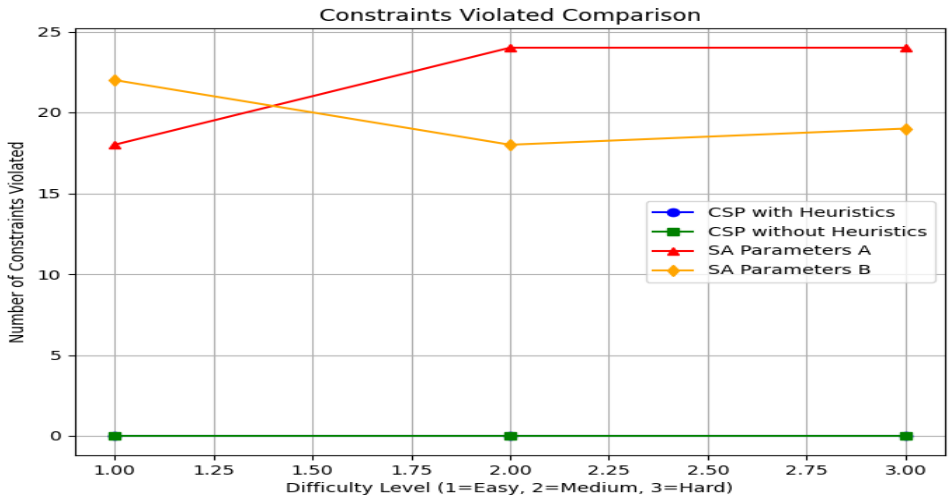

Figure 3: Experimental Results and Graphs - Constraints Violated Comparison (9X9)

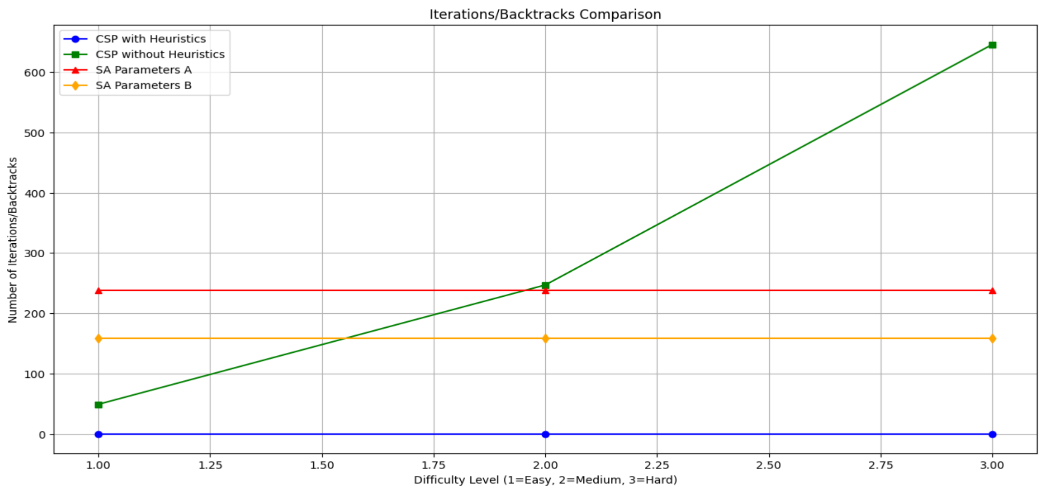

Figure 4: Experimental Results and Graphs - Iterations/Backtracks Comparison (9X9)

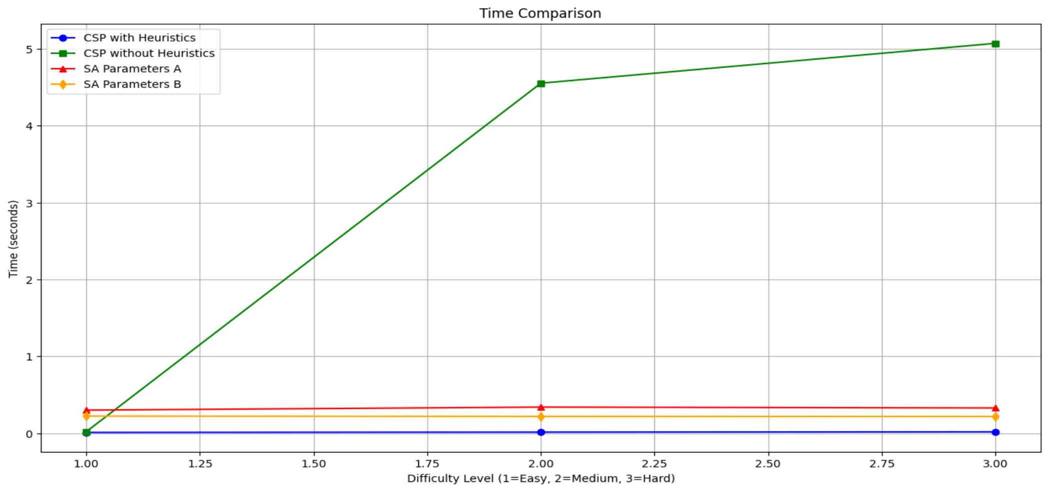

Figure 5: Time Comparison(16X16)

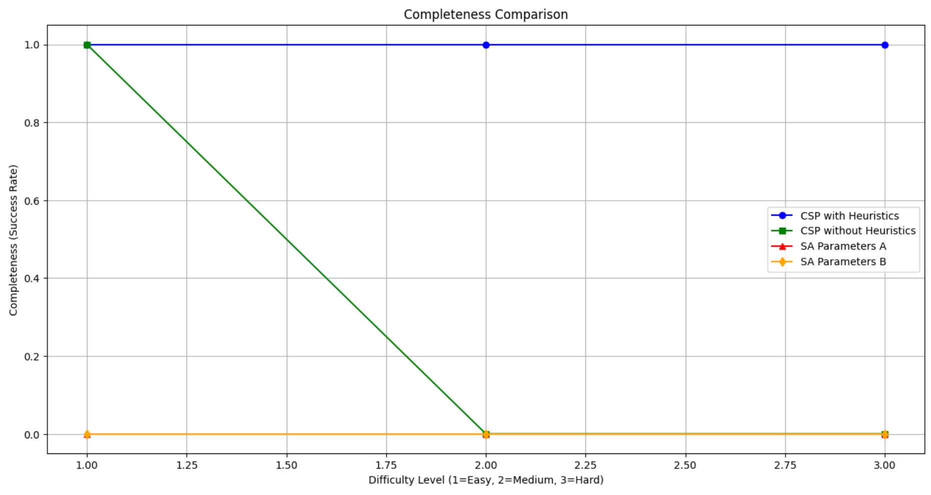

Figure 6: Completeness Comparison (16X16)

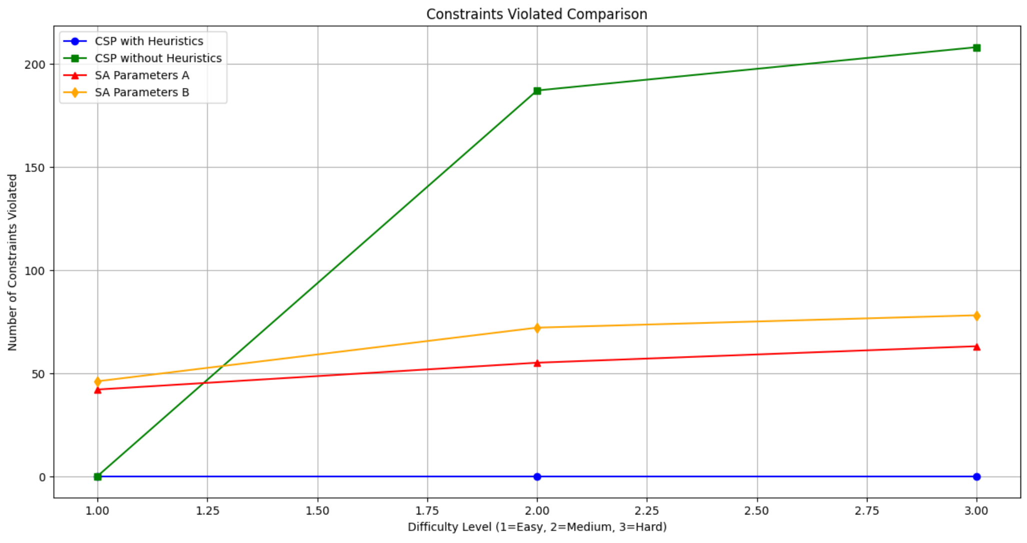

Figure 7: Constraints Violated Comparison (16X16)

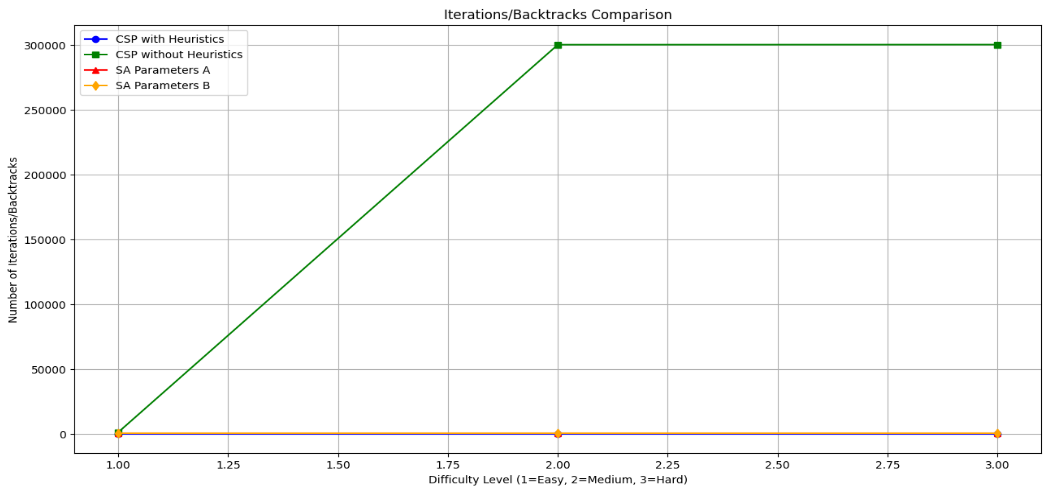

Figure 8: Iterations/Backtracks Comparison (16X16)

6. Comparative Discussion

1. Time: CSP backtracking with heuristic gives the fastest results in general for 9*9 and 16*16 sudokus, but it can sometimes choke on some harder problems with big size. Meanwhile Simulated Annealing gives a good result in reasonable time that may yield near-optimal solutions, on the other hand, CSP without heuristics is usually a bad choice especially for bigger and harder problems as it takes so much time to find a solution.

2. Completeness of Solution and constraint violated: CSP Backtracking ensures optimality in general, but since it can take so much time with no heuristic, it won't be completed for that time espically for hard problems, Simulated Annealing will provide a non-complete solution given certain parameter. The quality of SA improves with better parameter tuning, such as cooling rate and initial temperature.

3. Iterations/Backtracks: CSP backtracking without heuristics requires a huge number of backtracks as the difficulty increases, and for medium and hard puzzles it quickly reaches the capped limit of 300,000 backtracks, which reflects how badly the search space blows up without pruning. On the other hand, CSP with heuristics keeps the backtracking count very low across all difficulty levels because MRV/MCV/LCV with forward checking eliminates many wrong branches early. Simulated Annealing does not backtrack; it performs iterations instead, and even though it looks close to zero in the figure due to the scale being dominated by the 300,000 cap, it still keeps iterating within its budget and improves gradually, with different parameter settings leading to different convergence speed and solution quality.

7. Conclusion

In This project, two AI techniques were compared for solving Sudoku puzzles: CSP Backtracking and Simulated Annealing (SA), across multiple difficulty levels and grid sizes (9×9 and 16×16), focusing on runtime, solution quality, and scalability. The results show that CSP backtracking with heuristics (MRV/MCV/LCV and forward checking) delivers the most reliable overall performance, achieving the fastest solving times in general while maintaining a very low number of backtracks, which reflects strong pruning and efficient exploration of the search space. However, despite its strong performance, heuristic backtracking can still struggle on some very hard large-size instances, where the search space becomes difficult even with pruning.

In contrast, CSP without heuristics demonstrates why heuristic guidance is critical; as difficulty increases, the backtracking count grows dramatically and quickly reaches the enforced cap of 300,000 backtracks for medium and hard cases, which leads to failure under the imposed limit and highlights the impracticality of pure backtracking for large or difficult puzzles.

Simulated Annealing provides a different trade-off; it typically produces results in reasonable time and scales more smoothly in large search spaces by performing iterations rather than backtracking, often yielding near-optimal solutions rather than guaranteed optimal ones. Its solution quality depends strongly on parameter tuning-especially the initial temperature and cooling rate-where better settings improve convergence and reduce remaining violations, while poor tuning can leave the final solution incomplete.

Overall, the study supports using heuristic CSP backtracking as the preferred choice when correctness and completeness are required, particularly for standard Sudoku sizes, while Simulated Annealing is a strong complementary approach when scalability and runtime practicality are more important, or when an approximate solution is acceptable (especially for larger grids such as 16×16). Future work can build on these findings by exploring stronger SA parameter-tuning strategies and hybrid approaches that combine CSP's deterministic pruning with SA's ability to escape local minima.