University of Puerto Rico
Mayagüez Campus
Mayagüez, Puerto Rico

Department of Electrical and Computer Engineering

CSP Algorithms

by:
Manuel Alejandro Umaña Rodriguez
Dahyna Gabrielle Martínez Pérez
Jan Luis Pérez de Jesús
Christopher Hans Mayens Matías

For: Profesor José Fernando Vega Riveros
Course: ICOM 5015, Section 001D
Date: May 5,2025

Abstract:

How can an AI agent solve complex problems that contain many constraints? Throughout the following report the objective is to study how different algorithms perform in a constraint satisfaction problem, given the same problem. The studied problem was the Zebra Puzzle which consisted of 5 houses in a row. In each house lives a person with a specific nationality. Considering that the owners have different preferences in candies, pets, and drinks, the goal of the puzzle is to match attributes with the houses and find out in what house does the zebra live and in what house do they drink water. For this particular environment four algorithms were utilized: Backtracking Search, Min-Conflicts, Arc Consistency 3 with backtracking, and Forward Checking with backtracking. The algorithms would analyze the constraints of the problem and attempt to obtain a solution. Results indicated that Arc Consistency 3 combinated with backtracking outperformed every other algorithm in both speed and efficiency, while Min-Conflicts struggled the most because of the tightly coupled nature of the problem's constraints. This project highlighted the strengths and weaknesses of various CSP algorithms and demonstrated the practical trade-offs between completeness, cost, and inference in algorithmic design.

Table of Contents:

Group Collaboration

The tasks amongst each team member were divided by each member's experience utilizing python, as follows:

- Jan Pérez, Dahyna Martínez, and Manuel Umaña were in charge of exercise 6.7 within the python code and its segments within the report.

-Christopher Mayens was in charge of the development of the report and presentation. In addition to the creation of the presentation with the help of everyone.

-All members of the group were included and participated in the video. Video editing was made by Jan Pérez.

I. Introduction

Some problems are so complex that simple local search algorithms do not suffice to solve them effectively. Previously, different types of algorithms were studied, such as local search methods like hill climbing and simulated annealing. However, these algorithms often struggle to solve problems that involve a large number of tightly coupled constraints. Such problems typically include multiple variables, interdependent constraints, and large search spaces, making brute-force or naive methods inefficient and impractical. To address these challenges, the Constraint Satisfaction Problem (CSP) framework offers a more structured and efficient solution approach.

Real-life applications of CSPs include scheduling delivery routes for logistics companies, optimizing resource allocation in manufacturing to minimize downtime, and managing healthcare operations by assigning staff shifts and operating room schedules [1][2]. These examples demonstrate the relevance of CSPs in industries where precision, consistency, and resource constraints are critical.

In our class, we studied the principles of CSPs through structured logic puzzles, particularly variations of the well-known Einstein's puzzle (also called the Zebra Puzzle). This classic problem illustrates the core components of a CSP: defining variables (such as house colors, nationalities, or pets), assigning domains (the possible values each variable can take), and enforcing constraints that dictate allowable combinations. By working through this puzzle using methods such as backtracking search, constraint propagation (e.g., AC-3 to enforce arc consistency), and heuristic strategies, we were able to understand how CSP techniques generalize beyond puzzles to broader problem-solving contexts.

This report aims to explore and compare different CSP-solving approaches, including backtracking search, backtracking with forward checking, backtracking with arc consistency, and a local search variant—min-conflicts. All of these algorithms are applied to the same instance of the Zebra Puzzle, allowing for a direct comparison of their performance and behavior.

Due to time and resource constraints, the scope of this project is limited to the Zebra Puzzle as a case study. The implementations are developed in Python and evaluated based on completeness and practical performance, rather than large-scale generalization or optimization. The goal is to illustrate the strengths and limitations of each CSP approach when applied to a moderately complex but well-bounded problem.

II. Related Work

Exercise 6.7 presented an adjusted version of the Zebra puzzle, more commonly known as the Einstein's Puzzle [3]. The puzzle goes as follows:

There are five houses in a row, each with a different color. In each house lives a person with a different nationality. The five owners all have different tastes in pets, candy, and drinks. Considering none of the preferences overlap, find where the zebra lives and in which house they drink water.

This puzzle represents an example of a Constraint Satisfaction Problem (CSP) and is the base for the development of this assignment.

The environment was determined to be single agent, deterministic, fully observable, episodic, discrete, and static [4]. The structure of the environment and constraints were known and understood by the agent. While this exercise was not episodic in the standard AI practice, each constraint could be treated individually but contributed to the final solution individually.

To solve this problem 4 different algorithms were utilized to attempt to solve the Zebra puzzle; these being: Backtracking Search, Min-Conflicts, Arc Consistency 3 using backtracking, and Forward Checking using backtracking.

Backtracking is a problem-solving algorithmic technique that involves obtaining a solution incrementally by trying different options and undoing them if they lead to a dead end [5]. It assigns values to variables one at a time and if a variable leads to some sort of conflict it "backtracks" the last assignment and tries again. This algorithm is efficient for small sized CSP's especially when paired with heuristics.

Min-Conflicts is a search algorithm commonly used in computer science to resolve constraint satisfaction issues [6]. Basically it starts with a random assignment and iteratively selects a conflicted variable and assigns it the value that results in the least amount of constraint violations. This process is repeated until a valid solution is found [6].

AC-3 or Arc Consistency is an algorithmic technique that ensures all binary constraints between variables in a constraint satisfaction problem are satisfied. The way it works is that it searches for a supporting value for each variable so that the constraints are met [7]. It removes values from variable domains that are inconsistent with neighboring variables. Once the domains are all filtered, backtracking search is applied on the smaller search space.

The Forward-Checking Algorithm is an enhancement of the backtracking algorithm that aims to reduce the search space by applying local consistency checks [8]. The way it works is that for each unassigned variable the algorithm keeps track of remaining valid values and once a variable is assigned a value, local constraints are applied. This eliminates inconsistent values from their domains. If a neighbor has no valid values left after forward checking the algorithm backtracks [8].

The performance measure utilized to study the effectiveness of the algorithm was not a simple percentage of success like done in previous assignments. Instead, multiple measurements were studied to be able to craft a well researched evaluation. The first metric one was the solution status indicating if an algorithm had successfully solved the puzzle or not. The second was the time, measured in seconds, that each algorithm took to obtain the best solution. Lastly, the number of steps taken to solve the problem was studied and recorded.

III. Performance Approach and Results

The implementation of the Zebra Puzzle integrates several artificial intelligence algorithms, each of which solves the puzzle using a different Constraint Satisfaction Problem (CSP) technique. The algorithms used include backtracking search, the min-conflicts heuristic, AC-3 combined with backtracking, and forward checking. These methods are directly imported from the AIMA[9] csp.py module and applied to a specific CSP instance called "Zebra_candy", which encodes the logical constraints of the Zebra Puzzle. The implementation creates the puzzle instance and then solves it using each algorithm individually, followed by a visualization of the results. To solve problem 6.7, the algorithms must determine in which house the zebra lives and in which house water is drunk. After each algorithm produces a solution, a graphical display is shown, presenting the attributes associated with each house. This visualization includes a water emoji to indicate the house where water is consumed and a zebra emoji to mark the house where the zebra is kept.

For this exercise the objective was to discuss different representations of the zebra puzzle problem as a CSP. To successfully obtain an answer, the previously mentioned algorithms were studied as they each possess a different level of abstraction that allows them to interpret the problem in different ways. Results showed that the Backtracking algorithm successfully solved the zebra puzzle problem. As seen in Figure 1, the backtracking algorithm managed to successfully understand the constraints and separate the attributes correctly. Figure 2 and Figure 3 showed that AC3 with backtracking and Forward Checking also were efficient in solving the zebra puzzle. Evidently there is not a Figure for the Min Conflicts algorithm. The reason for this is that this algorithm did not manage to consistently obtain a valid solution.

Min-conflict algorithm is the only algorithm that failed to find a solution, with 4.2799 seconds and 10000 steps respectively as seen in Table 1. The problem with this algorithm was the limit in steps as increasing the limit to 50000 would solve the problem 20% of the time. For the purposes of this experiment it would still be considered a failure as it would not be able to find a solution without using more resources. Backtracking was successful in finding a solution, although it was the worst in both time and steps taken for finding the solution with the time taken being 1.3732 seconds in 75586 steps, recorded in Table 1. The best solution found was when implementing arc-consistency across all the nodes before implementing the algorithm with the implementation having the best time and step out of all the backtracking implementations with 0.0289 seconds and 1684 steps respectively, as shown in Table 1. Lastly forward checking was also measured with this implementation being successful and having noticeable improvement on the first backtracking algorithm with it having 0.1614 seconds for completion and 4353 steps for completion, shown in Table 1. With these results it can be seen that making the problem arc-consistent leads to the best result with the algorithm backtracking, with

the second best result being the implementation with forward checking. A graphical representation of the results was created and displayed in Figure 4.

A. Figures and Tables

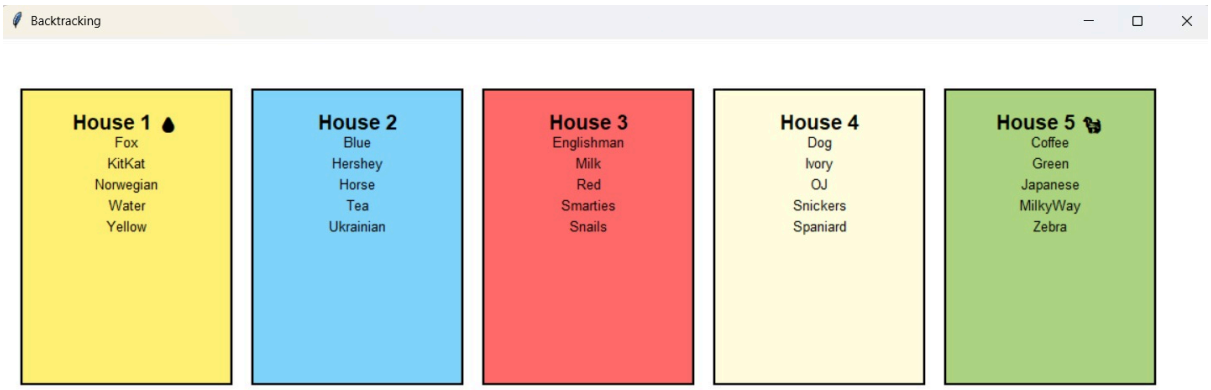

Figure 1. Zebra Puzzle solved state utilizing Backtracking Algorithm

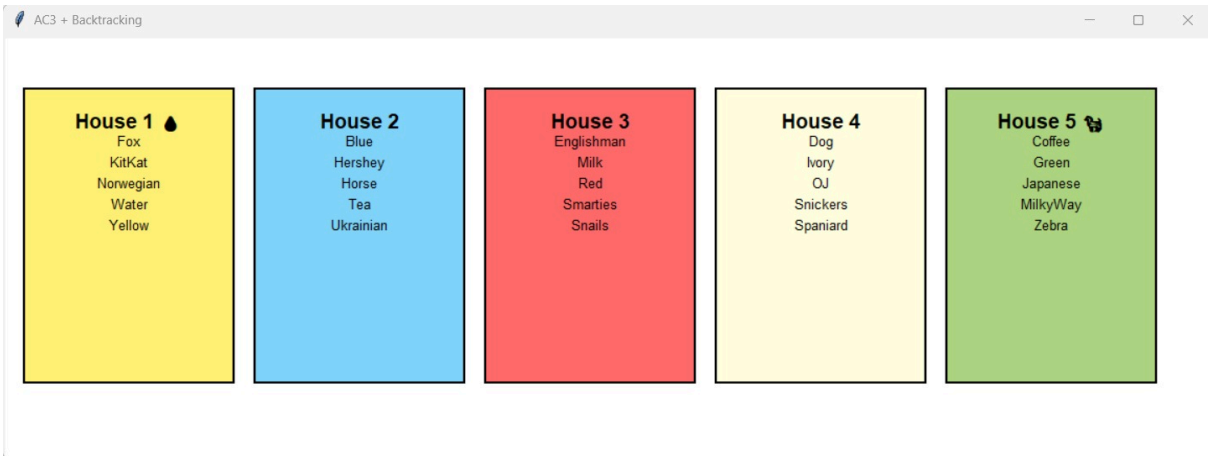

Figure 2. Zebra Puzzle solved state utilizing AC-3 + Backtracking Algorithm

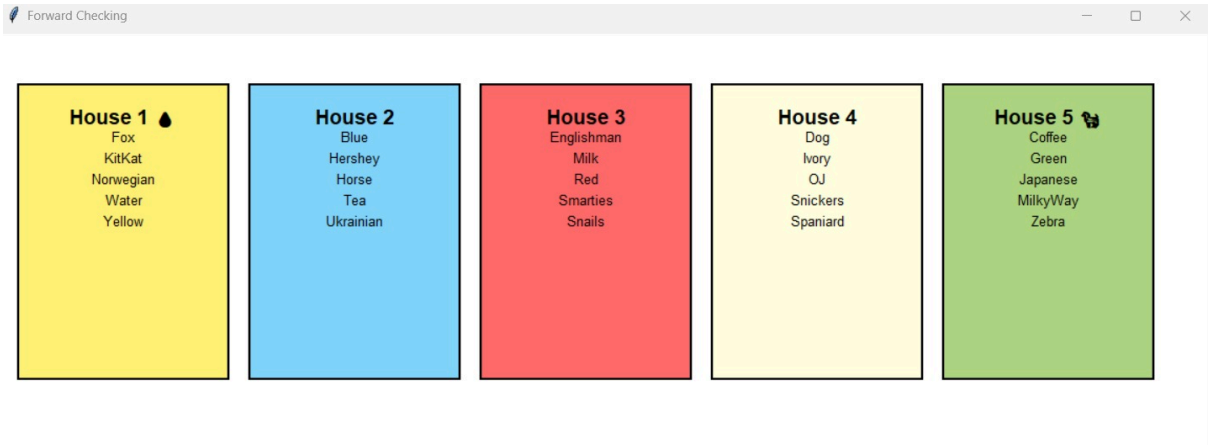

Figure 3. Zebra Puzzle solved state utilizing Forward Checking Algorithm

Algorithm	Status	Time (seconds)	Steps taken to solve the problem
Backtracking	Success	1.3732	75586
Min Conflicts	Failure	4.2799	10000
AC3 + Backtracking	Success	0.0289	1684
Forward Checking	Success	0.1614	4353

Table 1. Performance measurements of the different algorithms solving the CSP

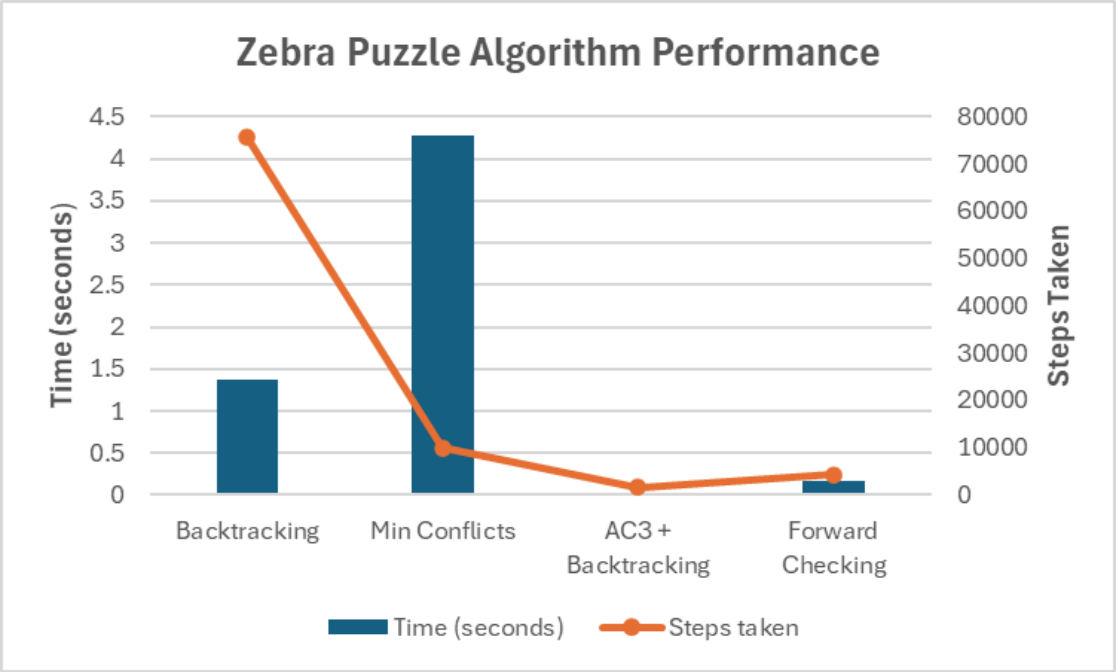

Figure 4. Performance measurements graph of the different algorithms solving the CSP

IV. Conclusion:

Throughout this report different applications of CSP algorithms were explored and categorized to find what algorithm was the best at solving 1 instance of the Einstein puzzle. This report has demonstrated the practical effectiveness of different problem-solving strategies in structured, constraint-heavy environments. By implementing and comparing standard backtracking, forward checking, AC-3 with backtracking, and the min-conflicts

local search method, we were able to observe how each algorithm handles the trade-offs between completeness, efficiency, and simplicity.

Among the algorithms tested, AC-3 with backtracking constraint propagation proved to be the most robust and consistent, successfully pruning the search space early and reducing the number of recursive calls required to reach a solution. Standard backtracking, while complete, was computationally slower and more prone to exploring irrelevant paths due to the lack of inference. Forward checking offered a moderate improvement by preventing immediate conflicts, though it still required significant backtracking in some cases.

On the other hand, the min-conflicts approach, though efficient in problems with large domains and fewer constraints, struggled with the puzzle due to the tight and interdependent constraints. Its performance was inconsistent and highly sensitive to initial assignments and random steps, confirming that local search strategies are not always ideal for tightly constrained, highly structured CSPs.

In summary, this project not only highlighted the theoretical strengths and weaknesses of various CSP algorithms, but also provided hands-on insight into how these techniques behave when applied to a well-structured logic problem. These insights form a strong foundation for tackling more complex and real-world constraint-based problems in future applications.

References:

[1] S.Yan, “Solving Real-World Puzzles: Constraint Satisfaction Problems for Data Science & Applied AI (incl. FREE Stanford-Berkeley Resources),” *medium.com*, Apr. 13,2025. [Online]. Available: https://luluyan.medium.com/solving-real-world-puzzles-constraint-satisfaction-problems-for-data-science-applied-ai-incl-44af36cf48c5 .[Accessed May. 3, 2025].

[2] AI Term Glossary, “Constraint Satisfaction Problem (CSP),” *ai-terms-glossary.com*, May 2025, [Online]. Available: https://ai-terms-glossary.com/item/constraint-satisfaction-problem/ .[Accessed May. 3, 2025].

[3] Geeks for Geeks, “Puzzle | The Einstein’s Puzzle,” *geeksforgeeks.org*, Feb. 22, 2023. [Online]. Available: Puzzle | The Einstein’s Puzzle | GeeksforGeeks .[Accessed May. 3, 2025].

[4] Geeks for Geeks, “Types of Environments in AI,” *geeksforgeeks.org*, Aug. 5, 2024. [Online]. Available: Types of Environments in AI | GeeksforGeeks .[Accessed May. 4, 2025].

[5] Geeks for Geeks, “Introduction to Backtracking,” *geeksforgeeks.org*, Jun. 24, 2024. [Online]. Available: Introduction to Backtracking | GeeksforGeeks .[Accessed May. 4, 2025].

[6] N. Jeevanandam, “The Min-conflicts algorithm, Incremental learning, and k-means clustering,” *india.gov.in*, Nov 3, 2022. [Online]. Available: The Min-conflicts algorithm, Incremental learning, and k-means clustering.[Accessed May. 5, 2025].

[7] S.Edelkamp and S.schrödl, “Arc Consistency,”, *sciencedirect.com*, 2012. [Online]. Available: Arc Consistency - an overview | ScienceDirect Topics .[Accessed May. 5, 2025].

[8] Geeks for Geeks, “Constraint Satisfaction Problems (CSP) in Artificial Intelligence,” *geeksforgeeks.org*, May. 5, 2025. [Online]. Available: Constraint Satisfaction Problems (CSP) in Artificial Intelligence | GeeksforGeeks .[Accessed May. 5, 2025].

[9] Aimacode, “aimpa-python,” *github.com*, Feb. 1, 2016. [Online]. Available: https://github.com/aimacode/aima-python . [Accessed May. 4, 2025].

Code Repository:

https://github.com/Dahyna-Martinez/Agents-and-Task-Management