Homework 2 - CSP Solver

Enver Eren

01/11/2024

Introduction

In this report, I will describe the Python script I developed to solve Constraint Satisfaction Problems (CSPs). The solver can handle classic problems like the N-Queens, Map Coloring, and Cryptarithmetic.

General Description

The script is designed to generate problem files for different CSPs and solve them using backtracking, enhanced with various heuristics. It includes functions to create specific problem instances and parse input files to extract variables, domains, and constraints.

Key Components

- **Problem File Creation**: I wrote functions like `create_problem_file_1`, `create_problem_file_2`, and `create_problem_file_3` to generate problem definitions for N-Queens, Map Coloring, and Cryptarithmetic problems, respectively.
- **Constraint Parsing**: The `parse_input` function reads the problem files and extracts the necessary CSP components.
- **Constraint Class**: I implemented a `Constraint` class that represents individual constraints and includes methods to check if they are satisfied given an assignment.
- **CSP Solver**: The core of the script is a backtracking solver that can use optional heuristics like Minimum Remaining Values (MRV), Degree Heuristic (DH), Least Constraining Value (LCV), and Arc Consistency (AC-3) to improve efficiency.

Heuristics and Algorithms

- **Minimum Remaining Values (MRV)**: When MRV is enabled, the solver selects the unassigned variable with the smallest remaining domain size. This helps focus on variables that are most constrained. In the script, I achieve this by iterating over all unassigned variables, determining their remaining (legal) domain sizes, and picking those with the smallest size.

- **Degree Heuristic (DH)**: The Degree Heuristic can be applied on its own or as a tiebreaker after MRV. When DH is enabled, the solver selects the unassigned variable that is involved in the most constraints with other unassigned variables. This focuses on variables that are most connected to others.

  - **Applied Independently**: If MRV isn't used, DH selects the variable with the highest degree by counting how many unassigned neighbors each variable has.
  - **As a Tiebreaker**: If both MRV and DH are applied and there's a tie after MRV, DH is used to decide among the tied variables. The solver counts the number of unassigned neighbors for each and selects the one with the highest count.

  In the script, after identifying the candidate variables (either all unassigned variables or those tied after MRV), I examine their neighbor relationships. I count how many neighbors are still unassigned for each variable and choose the one with the highest count.

- **Least Constraining Value (LCV)**: When LCV is enabled, the solver orders the domain values for a variable based on how few choices they eliminate for neighboring unassigned variables. This prefers values that leave the most options open for future assignments. In the script, I evaluate each possible value for a variable and count how many values it would eliminate from the domains of neighboring unassigned variables. Then, I sort the values accordingly.

- **Constraint Propagation (CP) using AC-3**: When CP is enabled, the solver uses the AC-3 algorithm to enforce arc consistency before and during the search process.

  - **Before Search**: I apply the AC-3 algorithm to the initial variable domains to prune inconsistent values, simplifying the problem from the start.
  - **During Search**: Each time a variable is assigned, AC-3 is reapplied to maintain arc consistency in the local variable domains. This helps detect inconsistencies early, avoiding unnecessary paths.

Example Runs

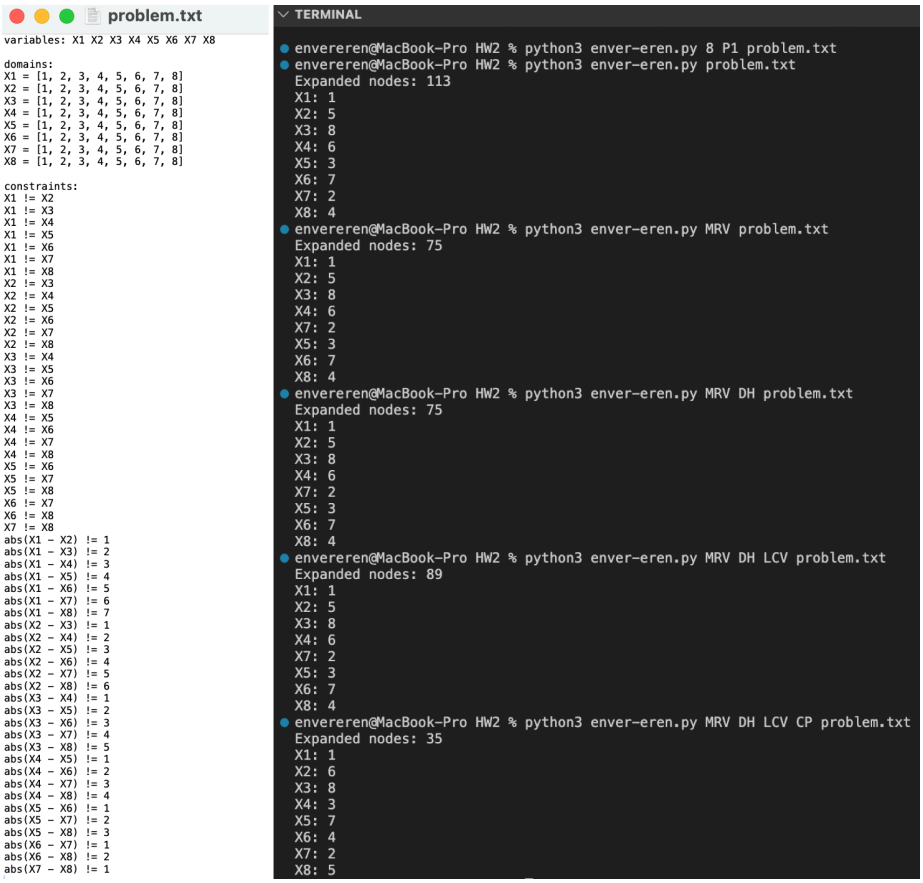

Figure 1: N queens problem with n=8

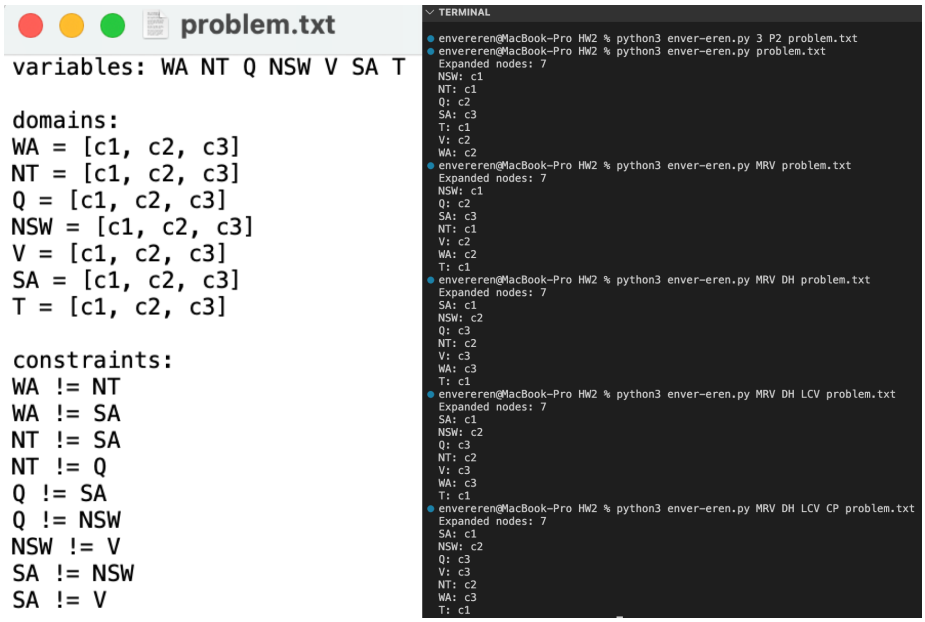

Figure 2: Map coloring with n=3

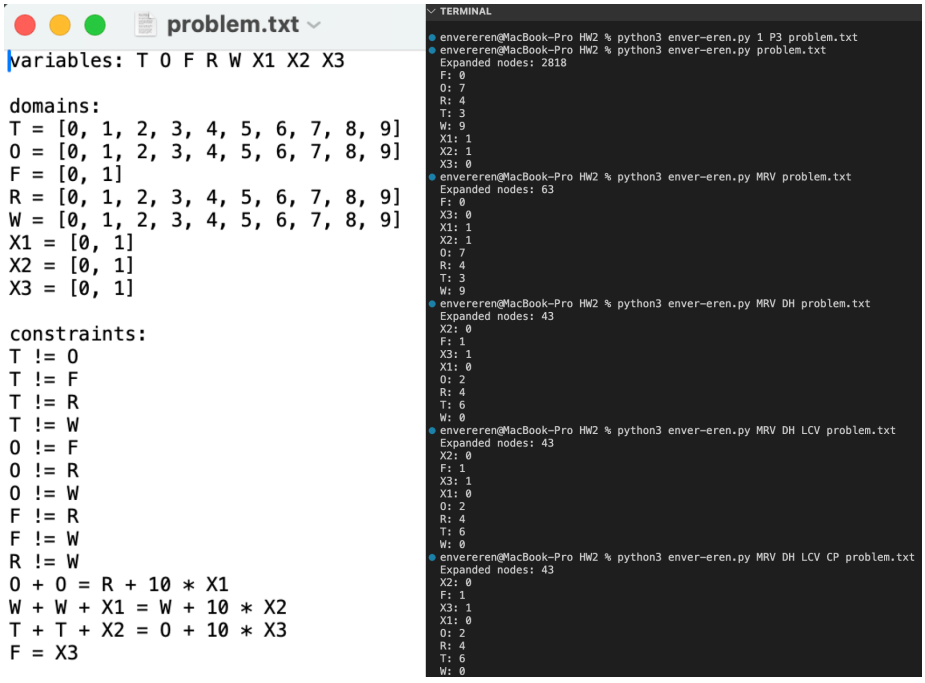

Figure 3: Cryptarithmetic with n =1