Minesweeper Solver

By: Nahom Amanuel, Irene Han, Andrew Hyssop Cha, Mia Ndousse-Fetter, Karthik Sellakumaran Latha

CMSC421: Introduction to Artificial Intelligence

Dr. Max Erlich and Dr. Sujeong Kim

University of Maryland College Park

May 5, 2024

Literature Review

Constraint satisfaction problems (CSPs) are used in artificial intelligence (AI) to solve combinatorial search problems such as scheduling, timetabling, type inference, and solving logical problems like sudoku or N-queens. Although its applications are diverse, a CSP is a type of search problem where a state is defined by some variables and domains. The objective is to find the values from the domain for the variables that follow the constraints (lecture notes). These problems have naturally been occurring in our day-to-day life for hundreds of years, but the classification and study of them has been relatively recent. In 1848 the N-queens problem was proposed by the chess player Max Bazzel, and it is believed that Theseus used backtracking search– a common methodology to solve CSPs– to move through the Labyrinth in ancient Greek mythology (Shultz, 2001). Development of modern CSP solving strategies emerged in the 1970's, but insufficient computational power held back progress (Brailsford et al., 1999). Later in the 1990's, there was enough renewed interest in CSPs that the *Constraints* journal was created in 1996, home to not only constraint programming and optimization, but also to machine learning, computer vision and other AI research topics (*Constraints*, n.d.). Strides have been made in the past decades in solving CSPs but many are NP complete including the subject of our project, Minesweeper. In 2000, Minesweeper garnered the attention of computer scientists when Richard Kaye published his groundbreaking paper, mathematically proving that the game is NP complete (Kaye, 2000). He showed that Minesweeper can be reduced to SAT (boolean satisfiability problem), a well-established NP complete problem, for any configuration of the game. Since this publication, researchers continue to explore how to reduce the computational complexity of difficult CSPs in various applications.

Introduction of Project

The Minesweeper game presents a player with a blank grid initially until a gray grid cell is clicked and what is underneath it and nearby cells is revealed. Some of these cells are blank, whereas others have numbers. The numbers represent how many mines are adjacent to the given cell, with the most being eight. If the player is certain about a specific mine location from the surrounding cell numbers, they can place a flag on that square. The goal is to uncover every cell that doesn't have a mine, and the game is immediately over if a player clicks and reveals a mine. Our project is a simulation of minesweeper that outputs a string representation of the board. A player can then use our Minesweeper solver tool that utilizes a CSP algorithm called backtracking search to determine which locations are mines and safe spots based on the current information available on the board. We define our constraints as the number of surrounding mines, our domains as O (safe), X(guaranteed mines), U (undecided), and our variables as unknown spaces surrounding the uncovered grids.

Initial goals and What Was Achieved

Our overall goal was to create a supplemental tool that minesweeper players can use to identify potential mines. It took some time to narrow down the scope to realistically complete the project on time. Initially we wanted to create a minesweeper simulator and solver– which we could easily change the implementation of our code to iterate through our search to do this– but we wanted our solution to be interactive. Throughout our testing, we found it to be more fun as a player to have the solver integrated into the game itself. We hope that users of our minesweeper solver also enjoy this element of interactivity. We would also like to note that we are solving a simplified version of the minesweeper game without flags to reduce the scope of the problem.

How Solution Works and Development

The beginning stages of our implementation was conducting a lot of research on how to practically implement CSPs in Python as well as meticulously defining our constraints, variables, domains, and the type of search we would use. Through our research and what we learned in lecture, we first decided on using a backtracking search with a filtering mechanism to assign variables. Backtracking can be implemented iteratively or recursively, but we decided on the recursive approach because it was most similar to how we solved CSP problems in homework. We also needed to include a filtering step to make it easier for our backtracking algorithm to generate potential solutions, and we implemented this in our filter function where we identify safe spaces and guaranteed mines according to constraints. We first tried a more traditional method where we found conflicting assignments and propagated this to other variables, but we found this process to be convoluted compared to assigning variables that clearly already satisfied the constraints. If we know that the values of a surrounding cell are all 0, then the cell must be a safe space, and if the number of total constraints is equal to the amount of variables then it's definitely a mine. Once we had a general idea of how to go about filtering and backtracking recursively, we designed our classes for variables, constraints and the csp as a whole. In tandem with the CSP development, we also created a minesweeper simulator that would give a string representation of the board. Our idea was that the string representation would be the input to our CSP algorithm which would give the user an updated board description of safe spaces denoted by "O" and certain bombs "X". The simulation was helpful in testing as we could directly use these string representations and the output of our CSP to see if our solutions worked. We then

implemented a UI that was able to take the output of our CSP algorithm and present it to the player that was easy to understand.

Results and Evaluation (Strengths, Weaknesses, Limitations)

We were able to connect the results of our CSP backtracking algorithm to our UI that identifies the likelihood of potential bombs to the player. Our solution uses our simulation to create a random minesweeper board that the user can play, and our solver that the user can use to win the game is powered by our CSP. A challenge we faced was determining what to do with cells that could be either a safe space or a bomb, as due to the amount of constraints the algorithm wasn't certain of either. We didn't want to add another recursive call to the backtracking, so we left it to be randomly chosen. This leaves a lot of room for potentially inaccurate predictions of a bomb or safe space. If we were to develop this project further, implementation of different heuristics and preprocessing techniques like least constraining variable and arc consistency could have helped decide what to do with these uncertain cells.

During our evaluation process, we noticed that our simulation and solving tools work very well on smaller boards that are around 10 by 10 tiles and performance declines on larger boards greater than 100 by 100 tiles. However, boards this big are seldom found in versions of Minesweeper. The tiles would also be too small to fit on the screen of an average computer, making it very impractical to play. To evaluate the performance we ran the simulation and solver and won 56% of games when selecting unmarked tiles (which are tiles that are undiscovered that the CSP did not mark as a high likelihood of having a bomb) given that the first chosen tile was not a bomb. In this situation we used a 10 by 10 board with 10 bombs. We believe accuracy can be increased by implementing more preprocessing and heuristics that were mentioned earlier.

Group Member Contributions

Nahom: Coding user interface, debugging
Irene: Coding CSP solver, debugging
Andrew: Coding minesweeper simulator, debugging
Mia: Coding CSP solver, debugging
Karthik: Coding CSP solver, debugging

Sources (APA format)

Brailsford, S. C., Potts, C. N., & Smith, B. M. (1999). Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, *119*(3), 557–581. https://doi.org/10.1016/S0377-2217(98)00364-6

Constraints. (n.d.). SpringerLink. Retrieved May 5, 2024, from https://link.springer.com/journal/10601

Kaye, R. (2000). Minesweeper is NP-complete. Mathematical Intelligencer, 22(2), 9-15.

Shultz, T. R. (2001). Constraint Satisfaction Problems—An overview | ScienceDirect Topics. https://www.sciencedirect.com/topics/computer-science/constraint-satisfaction-problems