1 Minesweeper

Minesweeper is a game played on an $n \times m$ grid. Try playing it online\(^1\) if you’re new to it! Formally, each cell $(i, j)$ on the grid either does or does not contain a mine. The game begins with each cell being *covered* – this means that the contents of the cell are not displayed to the player. The player moves by selecting a covered cell, which reveals the value of that cell and possibly others. For each revealed position, a value is shown, indicating the number of adjacent cells (up to eight) which contain a mine. The absence of adjacent mines is typically indicated by a revealed but numberless cell, but this is formally just a zero. The player loses if they select a cell with a mine. The player wins when all non-mine cells (called *clear*) are revealed. In our version of Minesweeper, the total number of mines is not known.

Here are some example Minesweeper boards. On the left all the clear cells are uncovered and the mines are visible. On the right some cells are covered and some are revealed.

Consider the reasoning process associated with a partially revealed board. The player knows the values of some but not all variables, and wishes to draw conclusions about those cells which are still unrevealed. Ignore for the moment how the user will actually chose amongst the various actions. We will now formulate Minesweeper as a constraint satisfaction problem. There are several possible formulations. Let there be two types of variables, $M_{ij}$ which indicates whether or not a mine exists at a position and $N_{ij}$ which indicates what number would be displayed if that position were revealed.

a) Draw the constraint graph for a 3x1 minesweeper board (which should have 6 variables).

\(^1\)http://gameswizard.com/j_ivmine.html
b) State precisely (but implicitly) what the constraint types in the CSP are. Assume that when a value of a variable is known, we encode that information using unary constraints.

Assume an \( n \times m \) board. For a spot \((i, j)\), if the value is known to be clear and contain the value \(k\), then there is a constraint \( N_{i,j} = k \) and \( M_{i,j} = \text{FALSE} \). For all \((i, j)\) there is the following constraint, \( N_{i,j} = \sum_{x \in S(i,j)} \text{mine}(x) \), where \( S(i, j) = \{(v, u) \mid 0 < v \leq n \text{ and } 0 < u \leq m \text{ and } (|i - v| = 1 \text{ or } |j - u| = 1)\} \) and \( \text{mine}(x) = 1 \) if there is a mine at \(x\) and 0 otherwise.

Now let’s consider choosing an action when faced with a partially revealed board.

c) Imagine we run a standard backtracking CSP solver on the CSP you defined above. We then pick a cell to reveal by taking the cells assigned as clear in the returned solution and picking one arbitrarily. Why can this approach result in a losing move? Explain briefly.

A CSP solver returns a possible configuration of mines consistent with the evidence. A square which is free in one satisfying configuration may be a mine in another.

d) In some cases, there will be moves which are guaranteed to be safe. How can we modify the standard backtracking DFS of a CSP solver to tell us which (if any) cells are guaranteed to be safe? Explain briefly, but precisely, what the key change in the algorithm would be.

One solution is to run the algorithm to make it try all variable assignments, rather than returning once it reaches a goal. Then it is only safe to move in a cell \(x\) if \(x\) is clear in all variable assignments. Another solution is to temporarily add the constraint \( M_x = \text{TRUE} \) (enforcing that \(x\) contains a mine). It will be safe to move to \(x\) only if the CSP with this added constraint fails to return any variable assignments. Actually, these two solutions are equivalent. The first enforces “for all variable assignments, \(x\) is clear”. The second enforces “there does not exist a variable assignment for which \(x\) is not clear”. These two statements are tautologically equivalent: \( \forall x f(x) \equiv \neg \exists x \neg f(x) \).

e) What might a reasonable procedure be for selecting a move when no move is perfectly safe? Explain briefly.

Multiple answers exist. One is to pick the spot such that it is assigned a mine in the fewest variable assignments (breaking ties randomly).

f) Why can we not use state-space search to plan a sequence of moves to win Minesweeper? Explain briefly.

Because we do not know for certain what board state will result from a given action, we cannot use deterministic search.
2 Scheduling with Arc Consistency

Alan is an ambitious undergraduate CS student about to begin his junior year. When planning out his classes, he realizes that he will be able to complete the four classes required to graduate in just three semesters. Imagine his excitement in learning that he will be able to graduate a semester early!

The four classes Alan needs to complete are: Advanced Algorithms (A), Bayesian Learning (B), Computer Programming II (C), and Distributed Computing (D).

There are some restrictions on these courses. Specifically: Advanced Algorithms (A) and Distributed Computing (D) are co-requisites (meaning they must be taken during the same semester); Advanced Algorithms (A) and Bayesian Learning (B) are always offered during the same time of day, and therefore cannot be taken simultaneously; Computer Programming II (C) is a pre-requisite for both Distributed Computing (D) and Bayesian Learning (B), and therefore must be taken in a preceding semester.

We will now formulate Alan’s scheduling problem as a constraint satisfaction problem in which each variable corresponds to a class Alan must take to graduate, and the values correspond to a numbering of his remaining semesters: 1 for the fall of his junior year, 2 the spring, and 3 the final fall.

a) State the domains of the variables and the constraints in mathematical notation using the following symbols: =, ≠, <, >, ≤, ≥. (Assume that each class is offered every semester.)

Let \( \{A, B, C, D\} \) be the set of variables. The domain of each variable is \( \{1, 2, 3\} \) which correspond to the semester in which he takes the class. The constraints are: \( A = D \), \( A \neq B \), \( C < D \) and \( C < B \).

We will now solve the constraint satisfaction problem formulated in (a) using backtracking search and arc consistency. Run arc consistency (AC-3) as a preprocessor and after each variable assignment. When there is ambiguity as to which variable to assign next, assign a value to the variable corresponding with the course that comes first lexicographically. Furthermore, when the next value to assign to a variable is ambiguous, break ties by assigning the earliest value first.

b) What are the remaining domains after enforcing arc consistency on the initial CSP with no assignments?

\[ A = \{2, 3\}, \quad B = \{2, 3\}, \quad C = \{1, 2\}, \quad D = \{2, 3\} \]

c) What are the remaining domains after assigning \( A = 2 \) and re-enforcing arc consistency?

\[ A = \{2\}, \quad B = \{3\}, \quad C = \{1\}, \quad D = \{2\} \]

d) State a solution to the CSP.

\[ A = 2, \quad B = 3, \quad C = 1, \quad D = 2 \]

e) True / False: With arc consistency, we will never have to backtrack while solving a CSP.

False