Introduction To Artificial Intelligence Homework Three Report

Andrés Ponce(0616110)

May 23, 2020

1 Background

For the second assignment of the course, we were tasked to model the game of Minesweeper as a constraint satisfiability problem, and apply suitable algorithms to solve a board given an initial setup. For the third homework assignment, instead of viewing it in terms of constraints, we model it as a problem of logical inference. Each cell in our board is then treated as a logical proposition which can either be true("safe") or false("mined").

Given this new approach, the task involved finding an assignment of the nodes which lead to a solved board in the end. Thus we treat a set of literals (or their complements) as a *clause*, which we can compare to other clauses to deduce their truth values. While this is not the most efficient way to solve a Minesweeper board, it does provide a useful insight into the area of logical satisfiability as applied to game-playing agents.

2 Program

The program is organized in a player module and a board module. The board module contains the information contained in every cell, the hint values and the types of the cells. The board module is just a container for the array containing the nodes.

We also maintain a clause class which, we use to store a list of nodes and their status. This is the main data structure we check when we want to compare the states two different cells. When the program starts, we initially insert the nodes marked as "safe", which are chosen at random from the board. There are then other considerations we make when mathing clauses or generating clauses from hints.

The player module is the one playing the game. This module is repsonsible for maintaining the knowledge base of the unassigned and the nodes whose status we have already determined. To do the matching, we try to match the node that is being checked with the other clauses that we have in our knowledge base.

The issues in this assignment come from the complexity of matching different propositions composed of nodes in our board. This introduced an extra layer of complexity at the moment checking a valid move. We have to match nodes based on interpreting them as logical propositions, which is not the most intuitive way of looking at a problem.

3 Further Discussion

3.1 How could we use first-order logic here?

First order logic differs from propositional logic in its power at expressing propositions. For example, a sentence such as "Every student will get a perfect score on this homework assignment" cannot be easily written using only propositional logic, unless we number all n students in the class and say the he or she will get a perfect score; something like $p_1 \wedge p_2 \wedge ... \wedge p_n$.

First order logic allows us to use universal or existential *quantifiers* which make a proposition apply to all the elements in the specified domain. Thus, if $g(x)$ represents the proposition "x will receive a perfect score on this assignment", we could model the previous sentence as $\forall x g(x)$, where x are the students in the course.

Since first-order logic already deals with applying predicate symbols to constant symbols, applying this structure to that of a program might prove more straightforward than using propositional logic. We could apply some functions to check the relations between the different objects. If we choose appropriate relations to check, maybe such to check a valid assignment or the number of adjacent objects, may be a more intuitive approach to this problem.

3.2 Is forward chaining or backward chaining applicable to this problem?

Yes, there are ways we could use chaining in this problem. To arrive at a conclusion c, we could recursively check all the propositions whose values we can infer. Special attention would be gven to inferring the value of the literals is c's premises. Backward chaining might be harder to implement since we would already need a solution to implement before inferring the values of its premises.

For forward chaining, the recursion tree would eventually encompass all the values that can be reached if we recurse. Once we build the tree, we could merely query the truth value of a literal whenever we required it. Thus recursively inferring values might prove a possible way to utilize forward chaining.

3.3 How could we imporve the success rate of "guessing" when you want to proceed from a "stuck" game.

3.4 How could we modify the method of Assignment 2 to solve the current problem?

We could modify the main algorithm from Assignment 2 to work in this assignment. For assignment 2, after checking if the board is solved, we checked if assigning the node to be a mine was a valid assignment based on the checking the adjacent nodes. In this assignment, we could attempt to select nodes and instead of checking its constraints, we could match it with our knowledge base to derive it's truth value. If we can do this for the current node, we can safely recurse on the adjacent nodes. If we cannot clearly match the node, we attempt to assign TRUE and FALSE to the current node and see which assignment leads to a solved board.

An approach resembling the above would still carry the same structure as the previous assignment, while using the techniques learned in this assignment involving propositional logic.