Ryan Barker

ECE 4420: Knowledge Engineering

September 19th, 2015

Takehome 1 Report

Section 1: Description of the Problem

The goal of this assignment was to use CLIPS to simulate moving a simple block moving problem. In this problem, blocks are moved across a game board containing singe 'S' and 'E' elements and multiple 'o' and 'x' elements. The block starts at the 'S' position and moves along 'o' elements until the puzzle is solved by reaching the 'E' element. 'x' elements block a path. Game boards can have multiple arbitrary solutions or no solution at all.

This problem required a suitable working memory representation in CLIPS be developed for blocks and game boards. The requested solution needed to be intelligent and use methods other than 'Generate and Test' to move the block. It also preferred minimal cycles where avoidable in the movement algorithm. The five 11x11 game boards given in the problem specification needed to be written into working memory by the program and solved upon CLIPS (reset) and (run) commands. The minimum required output for the program was a written and visual description of each solution.

Section 2: Background into the Intelligent Strategy of the Solution

To represent the game board from the problem in CLIPS, the engineer started with the board representation given in the problem specification and made three additions. First, the engineer added an integer 'board' slot, which was used to differentiate the cells in working memory for each game board. Second, the engineer added a 'B' member into the 'contents' slot of each cell to represent the block on the board. Lastly, the engineer added a yes or no 'adj_net' slot, which will be discussed below and is quintessential to the solution algorithm.

The developed solution to this problem involves analyzing each game board before making moves across them. It centers around something the engineer calls an "adjacency network" for each game board. An adjacency network is defined below:

Given a game board, G, the adjacency network for G is the set of all 'o' cells either adjacent to the 'S' cell or adjacent to an 'o' cell that is in the adjacency network. The 'E' cell may be added to the adjacency network if it is adjacent to an 'o' cell inside of the network. A solution to 'G' only exists if the 'E' cell is contained in the adjacency network.

Note the recursive nature of this definition. Developing the adjacency network for each game board allows the program to identify all valid cells that can contain the block, which in turn identifies the set of all valid moves it can make. This ensures the program always finds a solution to boards that can be solved and identifies boards with no solution. If a board has no solution, it can be discarded at this stage and the next board can be considered.

However, adjacency networks are only the beginning of calculating the best path on game boards with solutions. These boards can contain dead end cells, which are defined as follows:

A dead end cell is an ‘o’ cell that is either blocked in three or more directions or an ‘o’ cell that is blocked in two directions and adjacent to another dead end cell.

Dead end cells cause serious issues with avoiding cycles in the program. Picture a block on a board that has dead end cells. If the block ever enters a dead end cell, it must eventually cycle back into one or more cells it has already occupied to get to the ‘E’ cell. Obviously, the most efficient solutions to game boards contain exclusively forward motion. To more easily facilitate finding an optimal solution across a game board, dead end cells must not be considered for motion through it.

Once the adjacency network for a board is calculated and dead end cells are removed, developing a movement algorithm for finding an optimal path across a board becomes easy. The general idea is that a forward move toward an ‘E’ cell should always be preferred over another valid forward move, unless the block has to initially move away from the ‘E’ cell to get to the ‘E’ cell. This exception is exemplified in board five, where the block has to move north three times and east twice before it can move south toward the ‘E’ cell. This means that is safe to move toward the ‘E’ in a direction unless the last move was in the opposite direction. The decisions in this movement algorithm can be repeated until the block reaches the end of a board.

Section 3: The Implementation Itself

After some startup printouts, the program begins by placing a block on a game board by modifying the ‘S’ cell into a ‘B’ cell. At this stage, it asserts a fact containing the replaced ‘S’ so it can be put back later and a fact containing the board number of the board being operated on. This second fact is used throughout the program to ensure boards are only considered once at a time. This cell is also marked as being contained in the adjacency network (searched = yes), modified adjacency network (adj_net = yes), and in the final path (in_final_path = yes).

As written above, the next step in the program is to calculate the adjacency network for the board under analysis. This is accomplished with eight rules. Four of these start with cells marked with search equal to yes and check each valid movement direction (north, south, east, and west) for an ‘o’ that can be placed in the adjacency network. ‘searched’ and ‘adj_net’ are both marked as yes for all of these cells. The other four rules check for the ‘E’ off of the adjacency network and assert a solution_found fact stating a solution exists if they fire. All of these rules have a salience of 10 to ensure the entire adjacency network is calculated by CLIPS before moving on.

Next, the board is checked for checking for an absence of a solution_found fact. If this rule fires, all asserted facts are cleaned up and the next board is considered.

The third step in the algorithm, given solution_found has been asserted, is to calculate the modified adjacency network with all dead end cells removed. This is accomplished with five rules. The first four eliminate dead ends in each cardinal direction by marking adj_net as no for them. The final rule has negative salience and asserts a final_path_found fact once this is completed.

The final step in the algorithm is to move the block itself. This is accomplished with nine rules that only consider cells with adj_net equal to yes. The first four have elevated salience, but require the direction they move the block to be in the direction of the end to fire. The second set of four have regular salience and are for general movement when movement toward the end is not possible. When any movement rule executes, the destination cell is marked with in_final_path equal to yes and a previousMove fact is asserted to avoid cycles. The final rule in the movement algorithm checks for the saved symbol to be the ‘E’, which means the game has reached the goal state. It has the highest salience in the movement algorithm. When it fires, all facts asserted by the program are cleaned up and the next board is considered.

To print the board, a rule with a left hand side requiring a complete board and a print_board fact to be present was developed. This pretty prints the board, has the highest salience in the file, and retracts the print_board fact that caused it to fire upon completion. This way, any time the board needed to be printed after a rule (Initially and after each block move), a print board fact was asserted in that rule and the printing rule was automatically placed at the top of the agenda after it finished.

Section 4: Discussion of Results

As seen in board1.dribble, the program simply moves the block south on board 1 until the end is reached. Nothing notable happens here.

As seen in board2.dribble, the program moves the block as far north as it can go, then as far west as it can go, and finally as far north as it can go to the end. This behavior is a reflection of the CLIPS algorithm’s preference to move in a direction toward the end cell of the board.

As seen in board3.dribble, the program moves the block as far east as it can go, then south until it is adjacent to the end cell, and finally into the end cell itself. Note that the open cells south of the cell adjacent to the end cells are dead end cells and are not considered for movement.

As seen in board4.dribble, the program moves the block onto the start of the fourth board, but then states that the board has no solution and stops. The algorithm never looked into movement on this board, since it saw the end cell was not included in the adjacency network for it.

As seen in board5.dribble, the program moves the block away from the end cell to the north until it can cross over some open cells east and go south to the end cell. This happens because the open cells south of the end are unmarked from the modified adjacency network as dead ends. Note the program does not cycle north and south here, because of the measures described above.

Section 5: How good is my Solution?

The developed solution is effective. Because the program eliminates dead ends from paths, movement always occurs in a forward direction. Since it also ensures to avoid cycles and prefers to move towards the end cell of a board, it always selects a path with the minimum path length possible across that board. This is reflected in the results above, where paths with the minimum length are selected across all five boards.

There is definitely computation time present in the program, as it takes 433 cycles to complete. This number could probably be lower if steps of the algorithm were combined, but it would make the code less human readable. Also, a good portion of the rules being fired exist only to clean up asserts made by the program and take practically no time to execute. Overall, the engineer is quite happy with the solution he developed.

Section 6: Conclusion

If the engineer had the chance to start his development over, he would work on lowing the number of cycles the program takes to run by combining steps in the algorithm. He would set the program up so all path computations happen in one set of rules and all movement happens in another set of rules. He would also work on the solution to exclude corner cells from the possible paths, since he now recognizes they do not need to be considered for path generation. However, the engineer maintains that the solution he has presented is effective and suitable for the problem.