COMP 424 Final Project

Amy Hynes, Karla González

April 16, 2020

1 Technical Approach

The purpose of this project is to implement an AI algorithm discussed in class in the context of a two player zero-sum game. Our implementation is focused on optimizing the move our agent picks during each turn using a hill climbing algorithm. At every turn, each legal move is given a value which is used to rank its effectiveness at reducing the distance to the goal. More effective moves are assigned higher values than weaker moves. The highest valued moves are stored in a list, and once all moves have been evaluated, our algorithm randomly chooses a move from this list.

Each type of move has rules associated with how its value is determined. Playing a tile which moves the agent closer to the goal is preferred. The value is increased proportionally to the decrease in distance from the objective. Dead end tiles are never played as the value assigned to them is lower than other moves. Map cards are highly valued if the goal hasn't been revealed yet. Once the nugget location is known, the value for map cards is low enough that they are no longer played. Bonus cards are assigned a high value when there is a Malus card on our agent; otherwise, their value is 0. Malus is valued highly throughout the game, as the biggest factor in our losses is the Random Player placing a dead end tile in the path or destroying the path. As the number of turns approaches 50, or our distance to the goal decreases, the value of Malus increases further. Destroy moves are valued only for dead end tiles, and are more highly valued for dead end tiles further down the board (closer to the goal). Drop moves are valued low enough that if there is a more effective move it is played first. Dead end tiles are dropped before other cards because they are not useful. Map cards are dropped once the objective is revealed. Bonus cards are dropped if no other card is suitable to drop.

Hill climbing's efficiency depends on the evaluation function and whether it is computationally heavy. Our objective function is $f(n) = h(n)$ where $h(n) = max(x) \forall x \in X$ where X is the set of values of all legal moves in the current state.

On 1000 rounds of Saboteur, our agent won 300 and lost 84 with the remainder being draws. This is a win-loss ratio of 3:8.4 or a win rate of 78%.

2 Motivation for Technical Approach

The motivation for our approach is maximizing the value of the chosen move at each turn. This allows us to treat our choice of move as an optimization problem, where the goal is to find the best state (move to play) according to the objective function described in the previous section. Figure 1 illustrates a state-space landscape that we could obtain from the game [1].

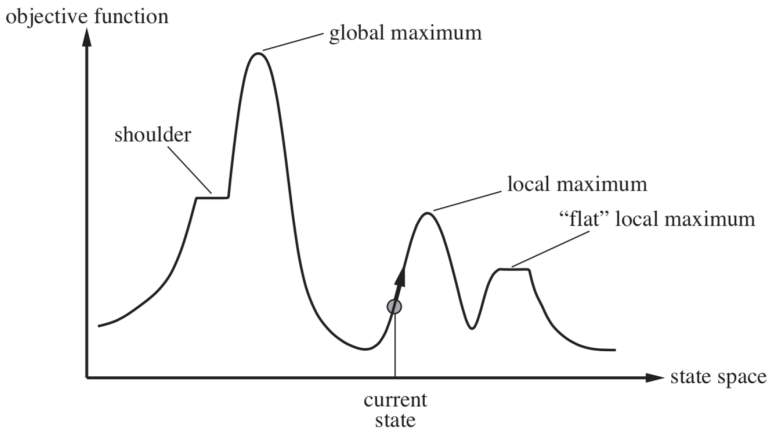

Figure 1: A state-space landscape has both "location" (state) and "elevation" (value of the objective function). The aim is to maximize the objective function by finding the global maximum at each turn.

Hill climbing is an appropriate strategy for this game as it chooses a move that maximizes the objective function by choosing a move with the highest possible value at each turn. Hill climbing does not look beyond the immediate neighbors of the current state, which means the space needed to implement this strategy is significantly smaller than other local search algorithms which store large sections of the game state tree. Note that hill climbing search has O(bd) space complexity where b is the branching factor and d is the solution depth. Hill climbing chooses the best neighbour of the current state without looking ahead and planning a strategy that spans multiple moves. This is why hill climbing algorithm is also called greedy local search.

3 Pros and Cons of Chosen Approach

The most important part of the chosen approach is the ranking system our agent uses to decide its next move. This objective function uses a scoring system which ranks our agent's potential moves. This allows the agent to differentiate between more and less effective moves, allowing it to play as efficiently as possible. The ranking system was designed to implement a structured way of how a human would approach the game with the objective of winning. By implementing this ranking system, we allowed our agent to approach each turn with a strategy which would enable it to reach a winning game state more efficiently than an agent with a random implementation.

The choice of the hill climbing algorithm comes with the benefits of a greedy approach. These algorithms usually perform well at optimization problems, but as with any approach, there are drawbacks. Hill climbing search may get stuck at a local maximum where it is drawn upward toward the peak but gets stuck with no potential moves as it only chooses upward moves. To overcome this potential obstacle, we designed our algorithm to only take into consideration the ranking of potential moves given our current state. We do not compare our neighbouring states to our current state, but rather just compare the neighbours to each other and choose the best possible move at that turn. Since the game is structured such that the agent gets a new card at each turn, planning one step ahead is sufficient. Though this approach may not result in the most efficient solution and is not guaranteed to be complete, it does mitigate the possibility of our algorithm getting stuck as a traditional hill climbing algorithm can. We must note that hill climbing algorithms are neither complete nor optimal.

4 Other Approaches

This assignment allowed for creativity in choosing an approach and as a result there is a long list of other potential strategies that could have been chosen. Initially, our approach was brute force. After having played the game a few times, we formulated a strategy that came intuitively to us as humans. We hard-coded the strategy and allowed our agent to play as follows:

1. Do not branch upwards - always discard that possibility
2. Search for downward movements and play them as often as you can, choose a move that minimizes the distance to the goal
    - Prioritize branching downwards rather than sideways
    - Branch sideways if
        (a) You do not have any tiles in your hand that make immediate downward progress
        (b) There are no active paths with exits on the bottom of the last tile on path; you may have downward cards but the game isn't set up to receive them
        (c) There are obstructions: there is a possible path downwards but there is another incompatible tile blocking the potential move
3. As soon as a map card is obtained, play if the gold has not been discovered and the opponent is not dangerously close to winning
4. Drop dead-end card if no other move
5. Save one bonus card in case struck by Malus
    - If there is more than one Bonus card in your hand, you may drop them until you only have one left

This became the basis for the objective function implemented in the final version of the code.

We also attempted to implement MiniMax search. The MiniMax algorithm computes the max value from the current state. It uses a simple recursive computation of the min or max values of each successor state. The recursion proceeds all the way down to the terminal states, and then the min/max values are propagated upward through the tree as the recursion unwinds. This algorithm was not implemented in our final code because it assumes that both players are playing optimally, and thus does not work when the agent we play against is acting randomly.

5 Future Improvements

The most intuitive next step to take would be to implement a check for dead end or destroyed tiles in the path and repair them to reduce the number of wins by the random player. This could be done by using a search algorithm starting from the entrance and traversing the paths that branch out from there. This would be done in parallel to the game and could increase our probability of a win.

We could also expand our hill climbing search to look more than one move ahead. This would also allow us to implement simulated annealing. This algorithm is similar to hill climbing but allows for some "bad moves" to try to escape local maxima, or generally stuck spots in the state-space landscape. This algorithm is careful, in the sense that the size and frequency (overall probability)

of random moves decreases over time. The probability of random moves can either decay to zero over time or a more complex implementation would be to have a value that depends on how "bad" the previous move was.

Bidirectional search could be used to improve the efficiency of building a path from the entrance to the goal. The idea would be to run two searches simultaneously: one as a traditional search would, forward from the initial state, and the other search would run backwards from the goal state. If the search is successful, they will meet in the middle. This search can also be implemented in a tree that has not been expanded completely by having the second search begin at the most optimal leaf. To implement this, we would need our algorithm to expand more than one level at a time as it does now.

References

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, USA, 3rd edition, 2009.