COMP 424
Project Report

Yi Tian Xu
260520039

April 11, 2014

Overview

The aim of the project is to implement an AI agent that can play halma on a 16×16 board with 13 pieces. To satisfy the constraint of one second between each move, searching the optimal action by exploring the search tree using, for example, Minimax, until a reasonable depth seems to me an infeasible solution. Considering the number of moves each of the four players can take, my program may have already ran out of time at depth two. Thus, my approach is to expand until depth one and use Monte Carlo, simulated annealing and a set of heuristics to approximate the best action.

The way the program is designed is inspired by RPG games. The implemented agent gains new tools to complete the game as it reaches closer to the goal. It starts by running a series of pre-selected moves while setting up its environment variables. Then it switches to Monte Carlo search with random policy and heuristics that will "level-up" according to the progress of the agent. When the chosen move is a "hop", simulated annealing is used to choose the next sequence of moves until the turn terminates.

Heuristics

Two types of heuristics are used: one describes the general positioning of the pieces (GPP); the other one specifies the improvement of a particular move (IPM). Each of them labels the board with three frontier: the longest diagonal osculating the base, the longest diagonal separating the base and the goal (LDS), and the longest diagonal osculating the goal. GPP counts the number of pieces in different sections while IPM add emphasis to the moves that can brings a piece from a section to another. IPM also takes into account the change in coordinate of the piece associated to the move - that is whether the piece ends at a position closer to the goal, or other players goal. In particular, it contains a function (MDD) that describes how much a move diverges from the longest diagonal that passes through the goal and the base (LDP). Since coordinates vary on the player ID, a rescaling function is needed to support these heuristics.

A few constraints are required to avoid some drawbacks that the heuristics bring. For instance, if a piece near the goal moving to goal results a higher weight than a piece moving out of the base, the program may focus more on the first pieces that get close to the goal. Thus, the function associated to the heuristics are carefully designed to handle different classes of states. In particular, the heuristics favour getting the pieces out from the base as soon as possible at the beginning of the game. When this is accomplished, these functions changes to encourage pieces to migrate to the other side of the LDS. This evolving property of the heuristics is what is similar to RPG player's level system.

The approximated value of an action is the sum of GPP, which is used to evaluate the resulting states given by Monte Carlo algorithm, and IPM, which evaluates the on the actions itself. However, for moves that are found using simulated annealing, their value is only based on IPM.

(See appendix for more information on the formula for the heuristics.)

Algorithms and Techniques

When the game start, a sequence of pre-selected moves are already given. This allows the program enough time to set up its variables and methods. Had this setup process run on the same turn with Monte Carlo, the agent would surpass the one second time limit. Moreover, since it is impossible for the other players to reach the agent's pieces for the first few steps at the beginning of the game, it is a safe assumption that 6 pre-selected sequences of moves can always pass the legality check.

Monty Carlo is packed with a pruning system that decrements or increments the number of rollouts when a move is found to be less or more promising. Currently, it is able to preform 10 rollouts, each with a length of 10 turns. The state at the end of each roll out is evaluated with GPP, which produces a weight generally between -3 and 3.

Simulated annealing is initialized with a temperature of 10 that decreases at a rate of 0.5 each time it is called.[1] It uses Boltzmann equation on the weight generated by IPM, which are generally between -5 and 5.

Both algorithms are referred from course notes.

To avoid moving back and forth, such as in the case when a backward hop is chosen by Monte Carlo, simulate annealing must remember to not move back to the initial position. Thus a private variable is maintained to store the previously selected action.

Results

This implementation is able to complete the game within 100 turns when playing against the random player or itself. The average number of turns is around 87 with lowest seen 74 and highest seen 93.[2] When playing against other agents (implemented by some other students), the number of turn may surpass 100 for winning games, but can still stays close between 85 to 95. Needless to mention further that the agent is able to complete each game without facing timeout. The speed of this implementation can be considered as an advantage for game playing AI agents that need to react fast.

[1] The temperature and rate are turned manually according to sample runs.

[2] These numbers are taken from a sample of 12 runs.

Comparing an implementation only with Monte Carlo and GPP, the addition of simulated annealing and IPM appears to bring down the average number of turn played by about five.

The main disadvantage of this approach lies within the "egocentrism" and the "greediness" of the heuristics. These may be a trade-off for faster speed; however, if the other players' state are taken into consideration, then the program may avoid choosing moves than can benefit more its opponents or obstruct its ally. Furthermore, MDD encourage the moves to align with the LDP. Although this creates a bridge near the goal, the pieces rarely take the bridge towards end of the game due the design of the MDD function. This brings another issue with MDD; the selected actions is often to move or to keep the pieces at the LDP even if there is a better sequence of moves next to them. In addition to the insufficient expansion of the first depth to find sequence of good moves that requires backward hop(s), this becomes a serious weakness of the agent. Yet, if the the MDD is weakened, the pieces are more likely to spread out towards the end of game, and to move one tile at a turn to reach the goal. Perhaps, this dilemma can be solved with a better approach.

Improvement and Future Works

A solution to avoid the disadvantages of MDD is to use depth-limited search instead of simulated annealing. Since the number of moves that follows a hop move is at most four, it is possible to push the horizon of the search tree further. This may allow evaluating the overall sequence of actions instead of each individual ones with MMD, so pieces are permitted to hop anywhere on the condition that they terminate near the LDP.

So far, the functions for the heuristics are manually chosen. If the constants in those functions were derived through machine learning, then the program may have a better performance. The starting sequence of pre-selected moves can also be learned, and a set of optimal moves can be used instead of the same ones for all games, provided a method to select the best starting moves according to, perhaps, the number of players that starts before or after the agent, or the starting moves of the other players.

Appendix: Formula

Given a state s with b number of pieces in the base, d number of pieces behind the LDS (that includes the pieces in the base), and g number of pieces in the goal,

$$GPP(s) = -L(b) - \delta_0(b)L(d) + \delta_0(d)L(13-d) + (1-\delta_6(d))L(13-g)$$

and

$$L(x) = \log(x * (1 - 1/t) + 1)$$

where t is the number of turn played. The delta function is defined as

$$\delta_y(x) = \begin{cases} 1 & \text{if } x \leq y \\ 0 & \text{otherwise} \end{cases}$$

This function helps to turn on or off some terms of GPP according to the progress of the agent.

Note that the log function is chosen specifically so that there's more importance given to a piece that is left behind alone than if it were with many other pieces. To adjust to the case when $x = 0$ ($\log 0 = -\infty$), the extra unit ("+1") is necessity. Furthermore, $(1 - 1/t)$ is chosen to increase the magnitude of GPP as the number of turn played grows.

Given a move $m = (x, y, x', y')$ where (x, y) and (x', y') are the starting and ending coordinate respectively,

$$IPM(m) = c' - c + C(c, c') \left(\delta_5(c)(1 - \delta_6(c')) \frac{t}{10} + \delta_{10}(c)(1 - \delta_{11}(c')) \log \left(\frac{t}{200} + 1 \right) \right.$$
$$\left. + \delta_{27}(c)(1 - \delta_{28}(c')) \log \left(\frac{t}{400} + 1 \right) \right) + MDD(m) + \delta_9(c) \frac{t}{10c}$$

where

$$c = x + y \qquad c' = x' + y' \qquad C(c, c') = \frac{|c' - c|}{c' - c}$$

and

$$MDD(m) = \frac{t}{280} \left(\frac{|x - y|}{2} - |x' - y'| \right)$$

$C(c, c')$ calculates whether the move is making a forward or a backward direction. The subtraction before that term $(c - c')$ gives the distance of the move. Note that $\delta_a(c)\delta_b(c)$ is detection whether a move is such that $c \leq a$ to $c' \geq 6$. The last term in IPM add a bonus for moves that starts from a position where $c < 10$.

MDD was initially designed to only give penalties when the ending position of a move is far from the LDP. The expression $|x - y|/2$ is added to also give reward for moves that starts further from the LDP. This version result a better performance of the agent.

The constants (i.e.: 10, 200, 280 and 400) are manually tuned based on sample runs.

Bibiography

Everything is from the course notes or from my head.