CMPS 241 Assignment - Hard—Impenetrable—Agent

Summary

We will use the Bitboard + Minimax + Alpha-Beta (BMAB) approach to ensure winning.

The degree of freedom in this model is the distribution of the agents' perfection levels.

Introduction

In this document, we will systematically explore an algorithm that will be, firstly, impenetrable, secondly, either minimizing the number of moves taken to win or maximizing the number of moves the opponent needs to take. Regarding win-ability, we will look into the most well-known efficient or non-efficient winning strategies. This will allow us to ensure the primary goal of the competition. Secondly, we will analyze these algorithms to see which is compatible with the second criteria. Considering we have a mathematically perfect winning strategy (if we use dynamic programming, but very good without), we realize that the ranking that the second criteria generates is dependent not only on our algorithm being passive, but also active. This calls for the possible use of game-theory to design an agent that can actively manipulate the scoring of other players: harming them by forcing them to make extra moves if it sees necessary to ensure winning rather than merely focusing on minimizing its own number of moves. It's still not clear whether this is possible, but we will explore this in this document.

Scoring

To ensure our work stays grounded with the scoring of the competition, we will set up a scoring system that reflects that of the competition, even if not directly. The following is the scoring method with n = number of games played and "loss" indicating 1 if the agent lost and 0 if it won.

$$\sum_{i=1}^{n} (-1)^{loss_i} \cdot 10^4 - gamemoves_i$$

The reasoning is as follows. Considering 50 students is an upper bound for the amount of competitors in the competition, suppose an agent plays, say, 2 times at most against every other one, the lower bound of the number of moves is 3 * (2 . 50) = 300, and the upper bound is 21 * (50 * 2)= 2100. Because of this, we can use the score of 10000 (>2100) safely as a score for winning, since if we play in the worst possible way but still get more wins we will win overall. The scoring thus measures two dimensions independently, the number of wins (higher priority) and number of losses (lower priority.)

Winning Strategies

Even if a strategy is optimal (plays perfectly), it cannot always win if the agent is the second player and the opponent plays perfectly. Agent 1 has a 100% win rate if it was perfect when it places its first disc in the center column (Victor Allis, 1988).

The Bitboard + Minimax + Alpha-Beta (BMAB) approach is the industry standard. No other strategy can beat this one. It will draw against itself (or P1 will beat P2), but it will never lose due to a tactical error. Bitboards allows the program to calculate millions of positions per second, which is necessary to reach the end of the game (depth 42) or deep enough to see the winning line. The total number of games is. *igrek51*'s solver handles over 4 million boards/sec on a regular laptop.

Agent 1 Position	Opponent	Outcome
First	Perfect Computer	**Agent 1 wins**
First	Human / Imperfect AI	**Agent 1 wins**
Second	Perfect Computer	**Agent 2 wins**
Second	Human / Imperfect Algorithm	**Agent 1 wins** (Once they make an error)

Table 1 showing best possible win-ability.

Game Strategies

We should care about the game-aspect of the match only if we play second. If so, we need to either. We shouldn't fall into the *Kamakazi Fallacy*. If we intentionally force the opponent to lose 20 points, we will lose 20 points equivalently. This will not affect the us vs them score since they won anyway; it only affects their own score with their opponents. Even if we later get enough wins to catch up with them, the difference made by our agent does not make a difference because of the equal loss. Equal loss is only loss to us, and only affects our competition with other players. This is especially true considering the agents have no information with regards to the competition state.

Let's analyze the possibility that the opponent is a fallible vs perfect statistically with regards to which strategy to pick (forfeit vs push for a win):

Let

p = probability a random opponent is perfect,

$1 - p$ = probability they are fallible,

w = chance we beat a fallible opponent when we "push for a win",

q = overall chance to win any given match under that strategy = $(1 - p) \cdot w$,

v = value of getting at least one win (the big ranking jump),

L = cost (loss, penalty, negative utility) per lost match if we play optimistically and resist losing quickly.

Assume we play matches until we either win once or stop after many trials. wins are iid with probability q each trial, so the number of trials T until the first win is geometric with mean

$$E[T] = \frac{1}{q} \quad (\text{if } q > 0)$$

The expected number of losses before that win is

$$E[\text{no. losses}] = E[T] - 1 = \frac{1 - q}{q}$$

Expected net payoff of the "push-for-win" strategy (stop after first win), then, is:

$$= v - L \cdot E[\text{no. losses}]$$

$$= v - L \cdot \frac{1-q}{q} \ldots (1)$$

Thresholds

If $q = 0$ (i.e. $p = 1$ or $w = 0$) then E[T] = ∞ and eq. (1) is $-\infty$, so pushing for a win is catastrophically bad when all opponents are perfect.

Further, if $q > 0$ then the expected cost to reach one win is finite. rearrange (1): pushing-for-win has positive expected value iff

$$v > L \cdot \frac{1-q}{q} \iff \frac{v}{q} > L \cdot \left(\frac{1}{q} - 1\right)$$

Intuitively: the one-time prize v must outweigh the expected accumulated loss before that prize arrives.

Corollaries that result:

1. If there is any positive chance of facing fallible opponents ($p < 1$ and $w > 0$ so $q > 0$), then as long as v is large enough relative to L and q, pushing for a win is the rational choice because we expect to pay a finite cost and capture the big prize. in particular, for tournaments where v is huge (rank jumps), even a small q can justify pushing for a win.
2. If all opponents are perfect ($p = 1 \Rightarrow q = 0$) then the formula says never push for a win; instead minimize losses per game. that formalizes "if it's inevitable we will lose, rush the loss" — quick loss minimizes $L \cdot$ (no. games) whereas resisting only increases expected cumulative cost.

Interpretation

The best possible strategy game-wise depends on the distribution of the perfection of the agents in the competition. If any of them are humans or potentially fallible, it's in our best interest to push for a potential win, as the expected score would be higher for this distribution, considering winning is better than any short term losses on the short term if the games went on for a long enough time. If

we know all other bots are perfect, we should push for a fast loss and have a higher chance of competing with other participants. If we don't know anything about the distribution, given a large enough number of games to be played, the benefit from potentially winning a single match is better saving any short term losses. In other words, winning a single match completely changes the ranks and space in which our agent participates in.

BMAB Agent

As per the statistical analysis, we will use a state-of-the-art Bitboard + Minimax + Alpha-Beta algorithm that is is fine-tuned to fit connect-4 with a number of optimizations mentioned in "report.pdf." In connect-4, we can represent the 7×6 board very efficiently using Bitboards: typically we have two 64-bit integers (or other fixed-size bitfields), one for each player, where each bit corresponds to a cell on the board. With clever encoding it's possible to only use 49 bits (7 columns × (6 + 1) "padding" rows) to represent all playable and sentinel bits. The extra bit per column helps avoid overflow issues when doing bit-shift checks. This representation allows super-fast operations: When an agent places a piece, we set a particular bit, check for wins by doing bit shifts + bitwise ANDs, and combine/inspect positions.

Minimax will be used as a search algorithm used to run over the game tree: from the current board position we recursively simulate all possible moves (with optimizations that make this computationally possible, discussed in the next paragraph), then counter-moves, assuming both players play optimally. In connect-4, there's a branching factor of up to 7 (since there are 7 columns to drop into), so the tree grows quickly. For leaf nodes (or when we hit a depth cutoff) we evaluate the board using a heuristic.

Alpha-Beta pruning make Minimax possible with our constraints. We cannot compute 2^42 which is in the order of 10^{12} and thus we need to cancel certain branches with our knowledge on the game and make certain compromises that make it computationally possible. We recursively search moves, maintaining two bounds, *alpha,* the best value the maximizer can guarantee; and *beta,* the best the minimizer can guarantee. Whenever the algorithm sees that a branch cannot possibly

improve on the current *alpha* or *beta*, it prunes that branch. Depth is limited to 11 based on our optimizations but it could be improved if we used a database built through bottom-up dynamic programming measuring a few gigabytes.

Pruning preserves the correctness of minimax by ensuring that the minimax value computed with pruning, v', is equal to the true minimax value, v, whenever $\alpha \leq v \leq \beta$ (with α, β initialized to $-\infty$ and $+\infty$). Full proof is found here.