Holey Ship! Bilging by reinforcement learning

Jesse Rodriguez, Tiffany Chen, and Jason Turner-Maier

December 12, 2008

1 Introduction

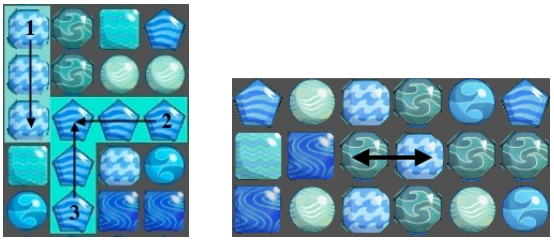
We constructed a Markov Decision Process (MDP) player of a Bejeweled-like game called "Bilging"– this game is part of a massively multiplayer online game based on puzzles. In this game, a player is presented with a grid of pieces (the board), and for each move he is allowed to swap horizontally adjacent pieces. If, after a move, three or more adjacent pieces of the same type are aligned in a row or column, the player gains some number of points based on the number and configuration of these pieces. Immediately, the pieces are then "broken" and the pieces below them are shifted up to fill the space. Random pieces are added at the bottom to fill up the resulting empty space. The scoring is such that breaking more pieces in a single move is worth significantly more than breaking them separately, i.e. breaking six pieces at once is worth significantly more than breaking three pieces twice. There are several different possible types of breaks based on the number of distinct horizontal/vertical matches made with one move. The player's overall score is determined by their average score per move.

There are several interesting features of the game that make it a compelling problem. First, the state space is large with 7e60 possible boards, so we cannot explicitly represent all possible boards. Second, a greedy strategy yields a poor average score, since the scoring function increases much faster than a linear function of number of pieces broken. Third, since the game can progress for an indefinite amount of time, we are solving for a problem that has no explicit final goal state.

The "bingo"

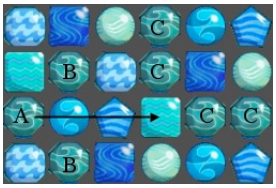
There are 21 types of matches (also called breaks) with large rewards for the larger break sizes. However, larger breaks are far less likely to occur by chance so they usually require several moves to reconfigure the board to make them. Human players are prone to making an irreversible mistake during these reconfiguration moves. It is generally agreed among the best Bilging players that the ideal trade-off between increasing the size of the break and difficulty of producing the larger break is what's called a "bingo," or a 3x3x3.

This type of break can be made often enough such that it is feasible to work towards it as a strategy, while also giving a large number of points for completion. Due to this point efficiency, we trained our Bilging player to prefer building bingos over other break types. If larger breaks are readily available, our player will take them, but bingos are the break type it actively works towards. As a point of notation, we will refer to a "one-away bingo" as a bingo that can be made making one move. For example on the left we have a matched bingo (3x3x3), and on the right we have a "one-away" bingo:



Blocker pieces

One problem that humans experience when making bingos is that they will move a piece towards completing a bingo, but the move will cause an unexpected break to occur near the bingo which destroys it. This is because nearby pieces are aligned to form a match and the move completes it. Because they hinder the player from making the intended bingo, we call these nearby pieces "blockers." Here is an example where a player is trying to complete a smaller match called a 3x3 but is hindered by blockers:

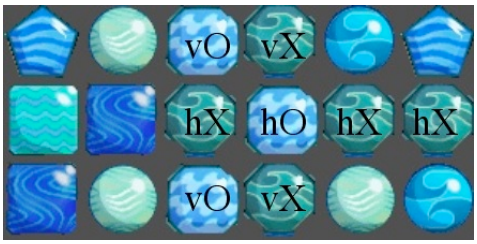


The player would like to move piece A over to the right to complete the 3x3 with the C pieces. However, if it does so in a naive fashion, it will form a break with the B pieces on the way and will be unable to complete the 3x3 since the A and B pieces will be destroyed. It needs to move one of the B pieces out of alignment with the other before moving A if it wants to successfully complete the 3x3. Therefore, the B pieces are blockers in this situation.

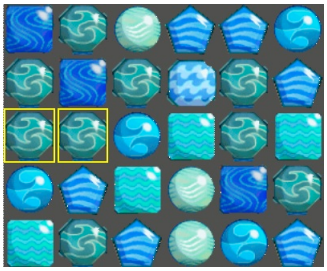
2 Methods

2.1 Features

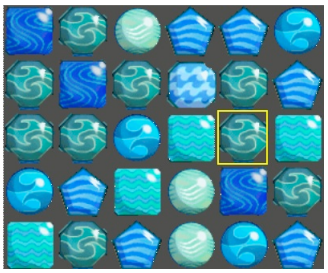
Since the state space of the problem is so large, in order for the learning process to be tractable we reduce our state space by representing boards as features of potential bingos on the board. We can easily detect if a bingo is possible by counting the piece colors present in each row. Let us make a few definitions before fully describing the features. For these purposes, assume a one-away bingo is structured as follows:



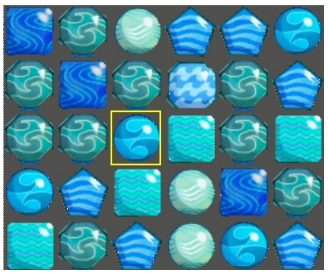
Here, the two color types are denoted as X and O. v and h refer to the row in which the piece appears. By convention, X refers to the more abundant color in the h row. Our features for a possible bingo measure how far from the one-away configuration it is. These features are defined as distances from the relevant pieces to a reference "anchor" point which is an estimate of where the hO piece should end up in the one-away bingo configuration. We use the following heurisic to guess where hO should be: first, we find the two closest X pieces in the h row:



Now, find we find a 3rd X piece in the h row, that is closest to the first two pieces. This 3rd piece determines the orientation of the bingo: if it is on the right, we call it a right bingo, if it's on the left it's a left bingo:



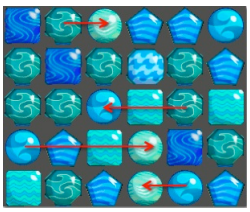
Then we set the anchor to be the position reached when walking one step from these two pieces in the direction of the 3rd:



Here are the bingo features we use to describe our boards:

- Sum of the distances from the two closest vX's not in the same row to the anchor
- Sum of the distances from the two closest vO's not in the same row to one away from the anchor in the direction of the bingo orientation
- Sum of the distances from the three closest hX's to the anchor
- Distance from the hO to the anchor
- Number of pieces which could not be moved naively into position due to blockers

These are features are depicted graphically here:



If there is no bingo possible on the board, all the features we describe above are set to their maximum possible values (eg the largest distance any piece could be from the anchor).

As an aside, we also tried many iterations of many different types of features before settling on these. They involved loose patterns and distribution of pieces on the board without specific breaks in mind. We found that training on these features either produced greedy behavior, or pursuit of the encoded patterns rather than matches that yielded long term high scores.

2.2 MDP Definition

States

Each state is a full board configuration which is a matrix with 12 rows and 6 columns with each element containing an integer value representing one of 7 colors. Formally, each state s is defined by:

$$s \in \mathbb{Z}^{12\times 6}, \quad s_{i,j} \in \{0, \ldots, 6\}$$

We do not consider boards with matching values to be valid states. Both of these statements hold:

$$\nexists\,(i,j) \text{ such that } s_{i,j} = s_{i,j+1} = s_{i,j+2}$$

$$\nexists\,(i,j) \text{ such that } s_{i,j} = s_{i+1,j} = s_{i+2,j}$$

Actions

Each action swaps two horizontally adjacent pieces such that taking action $a_{i,j}$ $(1 \leq i \leq 5)$ on state s causes exchange of values of $s_{i,j}$ and $s_{i,j+1}$.

Successor function

Our successor function $succ(s, a_{i,j}) = s'$ is deterministic if $a_{i,j}$ does not cause a match of 3 or more in a row such that s' is identical to s except that $s'_{i,j} = s_{i,j+1}$ and $s'_{i,j+1} = s_{i,j}$.

If $a_{i,j}$ causes a match, then s' is sampled randomly from distribution of successors states decribed in the dynamics above whereby matching values are removed, values are shifted vertically to fill their place, and the remaining values at the bottom are sampled with a uniform distribution over all 7 color values such that the newly sampled pieces do not cause any matches.

Reward function

We use a reward function $R(s, a)$ based on the current state, s, and the action a that is taken on this state. Any action that does not cause a match is given a reward of 0, and any action that makes a match is given a positive reward. We store the score given by each of the 21 of the match types (3x1, 4x4, 3x3x5, etc) in the vector c such that $c_k =$ score for match k with $|c| = 21$. So, if action a on state s creates a match k, then

$$R(s, a) = c_k$$

Value function

Since we use the state-action version of the reward function, we have to modify our value function from the state-only reward function described in the CS221 lecture notes[1]. We train our value function such that $V_\theta(s) = \theta^T\phi(s)$ approximates $\max_a R(s, a) + \theta^T \left(\frac{1}{k}\sum_{j=1}^{k} \phi(succ(s, a))\right)$.

2.3 Fitted Value Iteration

Training

We created a training set of 1000 boards where a bingo could be made with one move, and 1000 boards where a bingo could be made after several moves. We used a value of $\gamma = .99$ originaly but saw no behavioral difference from $\gamma = 1$. To estimate the optimal value function, we use standard fitted value iteration described in the CS221 lecture notes, except that we use the version that uses a reward function given by a state-action pairing such that at at iteration we calculate for each training example $\bar{s}^{(i)}$

$$y^{(i)} = \max_a R(\bar{s}^{(i)}, a) + \theta^T \left(\frac{1}{k}\sum_{j=1}^{k} succ(\bar{s}^{(i)}, a)\right)$$

Playing

After training we have learned the optimal value function V^*_θ, and in order to choose the best move to make at each state, the MDP player evaluates the expected value of each action by sampling each action k times. Thus it chooses to make the following action for state s:

$$\arg\max_a V^*_\theta \left(\frac{1}{k}\sum_{j=1}^{k} succ(s, a)\right)$$

This corresponds to a 1-move lookahead to optimize over our value function that guides it towards completing a bingo with each move.

2.4 Software

We used the aima-java package (http://code.google.com/p/aima-java/) for linear algebra subroutines for performing normal-equation-based linear regression during MDP learning. All

[1] http://www.stanford.edu/class/cs221/notes/Lecture10.pdf

other software was written by us, including the game simulator, the gui, the MDP, model training routines and, and game playing routines.

3 Results

By the numbers

When playing 50 moves per game, our MDP player obtains an average score of 8 points per move. This is much higher than a greedy implementation, which averages a score of 3.8 points per move. It obtains scores similar to above-average players (better than the authors of this work), but is significantly short of the best best human players; it is rumored that the top human players average approximately 12.0 points per move.

The MDP player completes 58%[2] of the possible bingos it encounters. While making moves towards bingo completion, it sometimes enters a local optima where it will decide not to make any more moves to complete the bingo. We consider this a failure of the MDP player since we trained it to complete as many bingos as possible. However, by comparison, a greedy player implementation achieved a 0% bingo completion rate over the same number of moves and games.

Behavior

This refusal to complete the bingo results from local optima in the estimated value function where it views any further moves around the bingo as worse than playing at any other position on the board. This situation tends to occur when the MDP player must swap two pieces where it moves one of them into a better position but the other into a worse one. Futher, it usually occurs when it is very close to the completion of a bingo.

It exhibits a couple interesting "intelligent" maneuvers that humans perform, and also some that humans are rarely capable of performing. Because we added features that take into account pieces that block the construction of a bingo, the MDP is able to recognize and move these blocking pieces out of the way. As mentioned before, human players can accidentally break a bingo before its completion, but it turns out that our player will actively destroy a posible bingo in order to optimize the rest of the board for another better bingo elsewhere which improves its overall score.

[2]Computed as number bingos completed divided by number of possible bingos encountered during 50 games of 50 moves per game

4 Discussion

We attribute the good (but less-than-perfect) performance to a couple factors. First, our MDP player's somewhat low bingo completion rate signficantly lowers its score. It seems to have the most trouble with very spread out bingos which have many pieces far from their correct position are difficult to assemble without falling into a local optima (as previously described). These same bingos tend to be difficult for humans as well. Having a one-move look ahead limits our value function in this regard since it is insufficiently expressive to create value function that is a strictly increasing while progressing towards a bingo in all possible cases. To address this problem, we may add a small search routine or a two-move lookahead which would help our player overcome these one-move local optima. Secondly, our MDP is not trained to optimally identify special cases where it can gain extra points by leveraging its knowledge of the scoring function.

Overall, however, the player does very well in constructing bingos and exhibits desirable behavior. In conclusion, we have produced a strong MDP Bilging player, and future work will need to focus to work on reducing the number of missed bingos to make it play optimally. Eventually, we aim to compete our player against a broader audience of human players.

Acknowledgements

We would like to thank Tom Do, Zico Kolter, and Ian Goodfellow for helpful comments, suggestions, and guidence in this project. The original Bilging game is made and owned by Three Rings Software as a part of their Puzzle Pirates product (http://www.puzzlepirates.com/).