Search Strategies in Multi-Player Drafting Games

CMPUT 651 Progress Report

Neesha Dessai, Richard Gibson and **Richard Zhao**
Department of Computing Science, University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
{neesha|rggibson|rxzhao}@cs.ualberta.ca

Introduction

Many commercial video games on the market today, particularly sports games, include some kind of drafting aspect. A draft works by having multiple agents take turns selecting from a collection of items until either agent has made a fixed number of choices or there are no more items left. For instance, when playing through multiple seasons of the baseball game *MLB 09: The Show* from Sony, a rookie draft takes place in between each season, where the human player and the computer opponents take turns choosing from a pool of new players who may then be added to the respective team's roster. Another example is the new hockey game *NHL 10* by EA Sports, where a fantasy draft can be activated at the beginning of a season. In a fantasy draft, all players are removed from their current teams, and then selected one-by-one by the agents (human or computer-controlled teams) in the league.

What is the best strategy for making selections in a draft in order to have the "best" set-up (or team of players) we could possibly achieve? We expect to develop a method, perhaps search-based or learning-based, which will result in "better" drafting than current methods used in our domain of choice, Risk.

Problem Formulation

We now formally introduce the problem of drafting. A *drafting game* is a finite, deterministic, full-information game played by a set of n players, which we label 0 through $n-1$. For simplicity, we assume that players' turns are in numerical order, so that player i's turn is always followed by player $i+1 \bmod n$. The game has a single set of actions or *picks* A initially available to every player. Once a player makes a pick a, that pick is forbidden to all players for the rest of the game. Play continues until all actions in A have been picked by the players. At the game's conclusion, the picks made by each of the players induce a partition $Z = \{A_0, A_1, ..., A_{n-1}\}$ of A, where A_i is the set of all picks made by player i. Each player then receives a reward signal $r_i(Z)$ according to the partition Z obtained. Player i's objective is to make picks in such a way to maximize $r_i(Z)$.

In many applications, a drafting game is a precursor to a larger competition involving the same set of players. For example, in Risk, players first participate in a drafting game, taking turns selecting territories until all territories on the map are occupied. The game then continues until a single player wins by occupying all territories on the map. When the drafting game is a prelude like this, it is sensible to model the reward signal $r_i(Z)$ as the probability that player i wins the "larger" game, given that Z is the result of the draft. We use this as our reward signal for our experiments with drafting in Risk.

Related Work

Perhaps the most widely known approach to computer game playing is the minimax adversarial search algorithm for zero sum games with two players, denoted Max and Min. The Max player (assumed to be the active player) performs a search of a game tree rooted at the current state of the game. Each leaf node of the tree is evaluated by a heuristic function, which returns a value estimating the merit of the state to Max. These values are then backed up the tree; at nodes belonging to Max, the maximum value of all child nodes is propagated up, whereas at nodes belonging to Min, the minimum value of the children is backed up. The Max player then chooses the action leading to the child with the maximum propagated value.

While minimax search is a traditional strategy employed in many two-player games, it is not applicable to games with more $n > 2$ players. Perhaps the simplest generalization of traditional minimax search to more players is the MaxN algorithm (Luckhart & Irani 1986). At the leaf nodes of the game tree, a heuristic function now estimates a vector of n merits $(h_1, ..., h_n)$, one for each player. At nodes belonging to player i, the vector with the highest h_i value is propagated up to the node. Thus, players are assumed to be maximizing their own individual payoffs throughout the remainder of the game. An alternative to MaxN is the Paranoid algorithm (Sturtevant & Korf 2000), where the active player assumes that the other players will attempt to minimize her payoffs with complete disregard to their own benefits. The Paranoid algorithm is essentially equivalent to the minimax algorithm, where the other players are represented as one meta-player (Min) attempting to minimize the individual heuristic value of the active player (Max). Finally, the MaxN and Paranoid propagation strategies can be dynamically selected according to the game situation. This is done in the MP-Mix algorithm (Zuckerman, Felner, & Kraus 2009), along with a third strategy called Offensive. However, MP-Mix was designed for games with only a single winner, which does not directly fit into the framework of a drafting game.

Monte-Carlo tree search algorithms, such as UCT (Kocsis & Szepesvari 2006), are a different approach to game playing than minimax-type methods. Simply put, thousands of games are simulated to completion, and each node's value is set to the average of the outcomes which passed through that node. As game trees typically grow exponentially in their branching factor, minimax-type algorithms must often rely on an accurate heuristic function at non-terminal nodes due to memory or time constraints. Monte-Carlo tree search, on the other hand, needs no heuristic function and receives an unbiased estimate of the value of each action. Because of this, UCT is often preferred in games with large action spaces or which lack a quality heuristic function, and has had much success in Computer Go (Gelly & Silver 2008). However, simple Monte-Carlo algorithms use random action selection during the simulations, which often lead to outcomes that will never occur among competent players. Minimax type algorithms, however, assume players behave in some logical manner. Our first contribution in this paper, MaxN-MC search, is a hybrid of these two approaches.

In drafting games, the initial action space is typically large. Rather than resorting to Monte-Carlo techniques, we can try to reduce the set of actions considered by the players to allow deeper minimax-type searches. One can automate this by using a tool called Genetic Algorithms with Meta-Models (GAMM) (Lee & Bulitko 2005). A simpler approach would be to consider only the most favourable actions according to the heuristic values of the immediate children. Our second contribution, the KthBestPick algorithm, incorporates this approach in a novel search strategy designed specifically for drafting games.

MaxN-MC Search

The MaxN-MC algorithm is an adversarial search algorithm for multi-player games. It applies MaxN search to a fixed depth in the game tree. Then, at each leaf in the search, we do not use a heuristic function to estimate its value as in the regular MaxN algorithm. Instead, we carry out Monte-Carlo simulations and average the outcomes to evaluate the leaf node. These outcomes are then what is propagated back to the root of the search in the remainder of the MaxN algorithm.

The pseudocode for MaxN-MC is presented in Algorithm 1. It receives the current state of the game as input, as well as two numbers, d and M, which denote the depth of the MaxN search and the number of Monte-Carlo simulations to perform at each leaf node respectively. First, MaxN-MC checks if the game is over (line 1), in which case then returns the associated payouts each player receives (line 2). For drafting games, we would return the vector of rewards $(r_1(Z), ..., r_n(Z))$ where Z is the partition of the initial action space as described previously. At a node at depth less than d, we make recursive calls on each of its children in the game tree, counting down the depth in the tree on each pass (line 3). Each call returns a vector of values, one for each player, indicating the merit of the child state to each of the players. The parent node then back-propagates the vector

Algorithm 1 MaxN-MC(Node $node$, int d, int M)

```
1:  if $node$ is terminal then
2:      return payouts associated with $node$
3:  end if
4:  if $d > 0$ then
5:      for all $child$ of $node$ do
6:          $child.value \leftarrow$ MaxN-MC($child$, $d-1$, $M$)
7:      end for
8:      Find $child$ of $node$ with greatest value for the active
        player
9:      return $child.value$
10: else if $d = 0$ then
11:     for $i$ from 1 to $M$ do
12:         Pick $child$ of $node$ at random
13:         $values(i) \leftarrow$ MaxN-MC($child$, $d-1$, $M$)
14:     end for
15:     return component-wise average of $values$
16: else
17:     Pick $child$ of $node$ at random
18:     return MaxN-MC($child$, $d-1$, $M$)
19: end if
```

with the best merit for the active player at this parent node (line 9). Once we reach a node at depth d, we perform M random walks down the game tree, each initiated at lines 12 and 13, and continued by lines 17 and 18 until returning a terminal value at line 2. Finally, for each player, we average the M merits associated with the outcomes of the random walks, and propagate back these averages (line 15).

The KthBestPick Algorithm

Theoretical Analysis

Empirical Evaluation

Conclusions

References

Gelly, S., and Silver, D. 2008. Achieving master level play in 9×9 Computer Go. In Fox, D., and Gomes, C. P., eds., *AAAI*, 1537–1540. AAAI Press.

Kocsis, L., and Szepesvari, C. 2006. Bandit based Monte-Carlo planning. In *15th European Conference on Machine Learning*, 282–293.

Lee, G., and Bulitko, V. 2005. GAMM: genetic algorithms with meta-models for vision. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2029–2036.

Luckhart, C., and Irani, K. 1986. An algorithmic solution of n-person games. In *AAAI-86*, 158–162.

Sturtevant, N., and Korf, R. 2000. On pruning techniques for multi-player games. In *AAAI-2000*, 201–207.

Zuckerman, I.; Felner, A.; and Kraus, S. 2009. Mixing search strategies for multi-player games. In *IJCAI*, 646–651.