

The Architecture of the Spewy Louie Jr. Poker Bot

Jon Parker

Department of Compute Science
Georgetown University
Washington DC, USA
jon@cs.georgetown.edu

Department of Emergency Medicine
Johns Hopkins University
Baltimore MD, USA
Jparker5@jhmi.edu

Abstract

A short discussion of the Spewy Louie Jr. No-Limit and Limit Texas Hold'em poker bot is presented. The hand clustering algorithm and the non-traditional game tree data-structure used are discussed in detail.

Introduction

Spewy Louie Jr. is a computer program designed to play both heads up (i.e. two player) Limit and No-Limit Texas Hold'em (LHE and NLHE respectively). Spewy Louie Jr. will be entered into the 2013 Annual Computer Poker Competition in the LHE and NLHE competitions. Spewy Louie Jr. represents a large scale refactoring of the prior code base. This paper is a short discussion of the current design and implementation. It should be noted that minor pieces of the design may change between now and when the competition software must be submitted.

Non-Traditional Game Nodes

The Spewy Louie Jr. architecture is designed around a non-traditional representation of the game tree. Nodes from the Spewy Louie Jr. game tree differ from nodes from a traditional game tree because they contain hand ranges rather than 2 exact hands (one for each player). The term "hand range" is borrowed from poker vernacular and is equivalent to a probability distribution over all possible hole cards. Defining a game tree this way helpfully reduces many of the state-to-state computations into simple matrix operations. However, defining a game tree this way ensures card removal effects are not perfectly accounted for at chance nodes. This noteworthy drawback is due to chance nodes only reflecting the current board cards rather

than the board cards and both players' hole-cards. Table 1 illustrates the difference between these two types of game trees.

Game Tree Chunks

The Spewy Louie Jr. architecture decomposes the game tree into easily solved chunks. A chunk of the game tree begins after the dealer has dealt new cards and ends when a player closes the action on any given betting round. Notice, this definition ensures that every chunk specific leaf node corresponds to a chance node. Processing a chunk involves 3 steps:

- (1) Build a small chunk of the game tree that reflects data stored in the global database including:
 - a. One hand range for each player at the chunk's root node. These hand ranges are computed from the immutable initial hand range and the saved player strategies.
 - b. A payoff matrix for each leaf node in the chunk. These payoff matrices are computed directly if a leaf node is a true terminal node (as in when a player folds). If a chunk's leaf node "hides" a downstream portion of the game tree then that payoff matrix is usually found in a database of previously computed results (if a previously computed result does not exist the matrix is estimated).
- (2) Compute an optimal solution as if the small chunk was the complete game.
 - a. Note: because the problem size is small a chunk's solution can be computed using linear programming or Counterfactual Regret Minimization (CFR).

Table 1: State Variables from two types of Poker Game Trees

| <i>Traditional Game Nodes</i> | <i>Spewy Louie Jr. Game Nodes</i> |
|------------------------------------------------------------|---------------------------------------------------------------------|
| double potSize | double potSize |
| Card[] boardCards | Card[] boardCards |
| Hand p1Hand | HandRange p1Range |
| Hand p2Hand | HandRange[] p2Range |
| double[][] playingStrategy //one entry for each child node | double[][] playingStrategy //one row for each bucket |
| double payoff (i.e. game value) | double[][] payoffs //payoff[i][j] = payoff when hands are (i , j) |

Initial experiments suggest CFR is still the superior approach because it yields a nearly optimal strategy quickly whereas the linear programming solution suggested by Koller, Megiddo, and von Stengel took significantly longer to reach a solution (by at least an order of magnitude). It should be noted that the LP package used for this comparison was the SimplexSolver in Apache commons.math. This package does not use a sparse matrix implementation. Consequently, the benefits of KMvS's technique were probably unrealized because their technique relies on sparsity.

- (3) Integrating the solution from one small chunk into the global solution.

Integrating the solution of a small chunk into a global solution is the only step that is not straight forward. As showing in Figure 1, a solved chunk produces two useful results: a payoff matrix (for that chunk's root node) and a playing strategy for each of the chunk's internal nodes. The payoff matrix is saved for later use. The optimal playing strategies extracted from a chunk are averaged into a set of preexisting playing strategies that correspond to those same nodes. Adding the "locally optimal strategy" together with the corresponding "sum of all prior locally optimal strategies" produces a revised strategy that is considered the current local playing strategy (after normalizing).

While unproven, it is the author's strong belief that consistently adding in locally optimal strategies will enable the global strategy to converge to a nash equilibrium. Of course, guaranteed convergence requires that all chunks from the game tree are processed. A proof showing how minimizing local regret also minimizes global regret can be found in Zinkevich et al. The author believes the logic in this proof can be easily transformed into something akin to "maximizing local optimality also maximizes global optimality" (provided a suitable definition of locally optimal is used).

Hand Bucketing

One popular technique used to reduce the size of the poker game tree is to group similar hands together and then play every hand within a single group the same way [Billings et al., Gilpin]. This technique saves space when storing the playing strategy because that strategy must be defined over m buckets instead of n hands. This technique also reduces the time required to execute an iteration of Counterfactual Regret Minimization (CFR) [Zinkevich et al. and Johanson] algorithm because the internal matrix computations occur on smaller matrices.

Spewy Louie Jr. currently distributes hands into 40 buckets (subject to change) except during preflop play. Hands are not bucketed in this later case. During preflop play all of the 169 different preflop poker hands are treated separately.

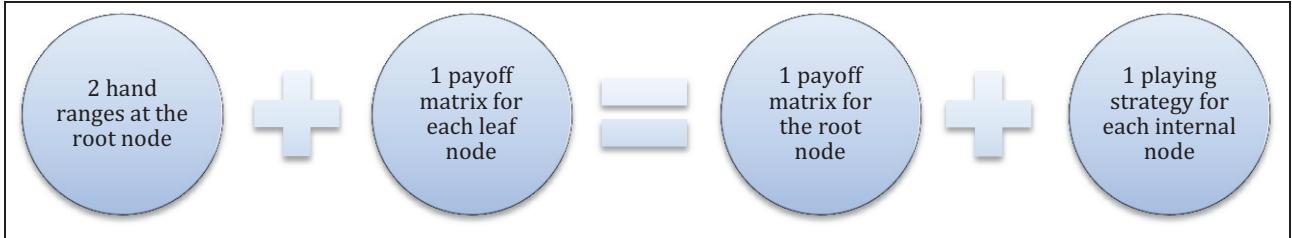


Figure 1: Computing the Solution to a Chunk: 2 inputs, 3 outputs

$$Sim(c_i, c_j) = \sum_{outcomes\ k} \left(\left(w_i (o_i(k) - a_{ij}(k))^2 + w_j (o_j(k) - a_{ij}(k))^2 \right) \cdot \max(o_i(k), o_j(k)) \right)$$

Equation 1: The Similarity between two hand clusters

To compute the hand bucketing for a given set of public board cards we begin by determining the set of percentile outcomes each possible hand can obtain by the time all the board cards are dealt out. For example, the set of outcomes associated with a strong drawing hand contains many values near 0 (for each way the draw can miss), many values near 1 (for each way the draw can come in), and a few middling values (for each way the drawing hand can hit a pair, but not the high value flush or high value straight). These percentile outcome sets are used to perform hierarchical agglomerative clustering on the hands that are possible given the current set of public board cards.

The metric used during agglomerative clustering emphasizes the differences between strong hands (or hand clusters). Let $o_i(n)$ be the n^{th} best outcome for hand i (or hand cluster i), $o_j(n)$ be the n^{th} best outcome for hand j (or hand cluster j), and $a_{ij}(n)$ be the weighted average of $o_i(n)$ and $o_j(n)$. The weighting process reflects the number of hands that have been assigned to each cluster of hands. Given this notation the similarity between two clusters i and j can be written as shown in Equation 1.

The first (additive) term in this summation represents the error introduced if hands i and j are merged into one entity. The second term weights those errors according to how strong the strongest source hand is. This style of bucketing is somewhat similar to the histogram based bucketing used in Gilpin, Sandholm, and Sorensen.

Overall Design

The Spewy Louie Jr. architecture stores two types of data: payoff matrices and playing strategies in highly concurrent lockers. The purpose of storing this data is to enable the revision of any particular chunk with as little prior computation as necessary. The payoff matrix locker saves payoff matrices for the root node of a chunk. When the payoff matrix of a leaf node needs to be accessed it is computed from the payoff matrices from the subsequent chunks (note: a leaf node is always a chance node). When the hand ranges of a chunk's root node must be accessed they are computed by "playing down" from the global root according to the strategies kept in the strategy locker.

In Development

Updating a chunk of the game tree requires knowledge of the payoff matrices that apply at that chunk's leaf nodes. In many cases these payoff matrices must be approximated. The best way to approximate these matrices is currently unknown. The current implementation returns the matrix resulting from assuming that no more betting will occur. It may be more efficient to find the "closest" situation to a particular leaf node and return the payoff matrix associated with that node.

References

- Billings, Darse, et al. "Game-tree search with adaptation in stochastic imperfect-information games." *Computers and Games*. Springer Berlin Heidelberg, 2006. 21-34.
- Gilpin, Andrew, Tuomas Sandholm, and Troels Bjerre Sorensen. "Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker." *Proceeding of the National Conference on Artificial Intelligence*. Vol. 22. No. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.
- Zinkevich, Martin, et al. "Regret minimization in games with incomplete information." *Advances in neural information processing systems* 20 (2008): 1729-1736.
- Koller, Daphne, Nimrod Megiddo, and Bernhard Von Stengel. "Fast algorithms for finding randomized strategies in game trees." *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. ACM, 1994.
- Johanson, Michael Bradley. "Robust strategies and counter-strategies: Building a champion level computer poker player." Masters Abstracts International. Vol. 46. No. 03. 2007.