Supplementary Materials for

Superhuman AI for multiplayer poker

Noam Brown* and Tuomas Sandholm*

*Corresponding author. E-mail: noamb@cs.cmu.edu (N.B.); sandholm@cs.cmu.edu (T.S.)

Published 11 July 2019 on *Science* First Release
DOI: 10.1126/science.aay2400

This PDF file includes:

Other Supplementary Materials for this manuscript include the following:
(available at science.sciencemag.org/cgi/content/full/science.aay2400/DC1)

Summary of technical innovations in Pluribus relative to prior poker AI systems

In this section we briefly summarize the technical innovations in Pluribus relative to prior poker AI systems. These techniques are discussed in more detail in the body of the paper or later in this supplementary material.

We split the innovations into three categories in the following three subsections, respectively. Unfortunately, measuring the impact of each of these innovations would be too expensive due to the extremely high variance in no-limit poker and the high dollar and time cost of conducting experiments against humans. We provide estimates of the magnitude of the improvement where possible.

Depth-limited search

Depth-limited search was the most important improvement that made six-player poker possible. It reduces the computational resources and memory needed by probably at least five orders of magnitude. Libratus (*6*) always solved to the end of the game when real-time search was used (Libratus started using real-time search on the third betting round—the half-way point in the game—and in certain situations even earlier). However, additional players increase the size of subgames exponentially. Conducting beneficial real-time search on the flop (the second betting round) with more than three players involved is likely infeasible without depth-limited search. DeepStack (*5*) used a form of depth-limited search in two-player poker, but the technique was already very expensive computationally in that context—because huge numbers of belief-contingent subgames needed to be solved in advance to obtain leaf values for the real-time search; this may not be feasible in six-player poker. We previously published a two-player

version of depth-limited search in imperfect-information games that is far less expensive in two-player poker than the real-time solving in Libratus or DeepStack (*41*). In this paper we generalized that approach to more than two players and included a number of additional technical contributions as follows.

- Our previous depth-limited search had the searcher play the blueprint strategy while the opponent chose among multiple continuation strategies. This is theoretically sound (in two-player zero-sum games), but in practice gives the opponents more power and therefore makes the searcher relatively defensive/conservative. The current paper addresses this weakness by having the searcher also choose among continuous strategies (which is still theoretically sound in two-player zero-sum games). Our prior search algorithm instead penalized the opponent to try to balance the players, but our new approach is more effective, easier, and more elegant.

- Previous nested search algorithms either used nested safe search or nested unsafe search (described in detail later in this supplementary material). Nested unsafe search is not theoretically sound, and in some cases may do extremely poorly, but on average tends to perform better in practice. We use a new form of nested unsafe search in this paper in which we always solve starting at the beginning of the current betting round rather than starting at the most recent decision point. Our player's strategy is held fixed for actions that it has already chosen in the betting round. However, the approach allows the other players to have changed strategies anywhere in the current betting round, that is, even above the current decision point. Taking that possibility into account mitigates the potential for unsafe search to be exploitable, while still retaining its practical benefits.

Equilibrium finding

Pluribus also has innovations in the equilibrium finding algorithms that are used in the blueprint computation and within the depth-limited search. Here we summarize them and we describe each of them in detail in the paper body or later in this supplementary material.

- We used Linear MCCFR rather than traditional MCCFR. While we already published Linear MCCFR recently (*38*), it was never implemented and tested at scale. We suspect this sped up convergence by about a factor of 3.

- Our Linear MCCFR algorithm used a form of dynamic pruning that skipped actions with extremely negative regret in 95% of iterations. A similar idea was used in Libratus and in BabyTartanian8 (our AI that won the 2016 Annual Computer Poker Competition), but in those cases the skipping was done everywhere. In contrast, in Pluribus, in order to reduce the potential inaccuracies involved with pruning, we do not skip actions on the last betting round (because on the last betting round one does not get the benefits of effectively increasing the card abstraction through pruning) or actions leading to terminal payoffs (because the cost of examining those payoffs is minor anyway). Additionally, whether or not to skip in Libratus/BabyTartanian8 was decided separately for each action rather than deciding for the entire iteration; the latter is cheaper due to fewer calls to a random number generator, so we do the latter in Pluribus. We suspect that these changes contributed about a factor of 2 speedup.

Memory usage

To conserve memory, Pluribus allocates memory for the regrets in an action sequence only when the sequence is encountered for the first time (except on the first betting round which is small and allocated up front). This is particularly useful in 6-player poker, in which there are many

action sequences that never occur. This reduced memory usage by more than a factor of 2.

Rules for no-limit Texas hold'em poker

No-limit Texas hold'em (NLTH) has been the most common form of poker for more than a decade. It is used, for example, to determine the winner of the World Series of Poker Main Event. Six-player NLTH poker is the most widely played format of NLTH.

Each player starts each hand with $10,000 and the *blinds* (the amount of money that Player 1 and Player 2 must contribute to the pot before play begins) are $50 and $100, so each player starts with 100 *big blinds*, which is the standard buy-in amount for both live and online play. By having each hand start with the same number of chips, we are able to treat each hand as a separate sample when measuring win rate.

NLTH consists of four rounds of betting. On a round of betting, each player can choose to either fold, call, or raise. If a player folds, they are considered to no longer be part of the hand. That is, the player cannot win any money in the pot and takes no further actions. If a player calls, that players places a number of chips in the pot equal to the most that any other player has contributed to the pot. If a player raises, that player adds more chips to the pot than any other player so far. A round ends when each player still in the hand has acted and has the same amount of money in the pot as every other player still in the hand.

The initial raise on each round must be at least $100. Any subsequent raise on the round must be at least as large as the previous raise (i.e., at least as large as the amount of money beyond a call that the previously-raising player contributed). No player can raise more than their remaining amount of money (which starts at $10,000).

The first round starts with Player 3 and each subsequent round starts with Player 1 (if Player 1 is still in the hand). Play proceeds through each consecutive player still in the hand, with Player 1 following Player 6 if the round has not yet ended. On the first round, Player 1

must contribute $50 and Player 2 must contribute $100 before play begins.

At the start of the first round, every player receive two private cards from a standard 52-card deck. At the start of the second round, three *community* cards are dealt face up for all players to observe. At the start of the third betting round, an additional community card is dealt face up. At the start of the fourth betting round, a final fifth community card is dealt face up. If at any point only one player remains in the hand, that player collects all the money that all players have contributed to the pot. Otherwise, the player with the best five-card poker hand, constructed from the player's two private cards and the five face-up community cards, wins the pot. In the case of a tie, the pot is split equally among the winning players.

For the next hand—i.e., the next repetition of the game—player 2 becomes player 1, player 3 becomes player 2, etc.

Experimental setup and participants

The players were allowed to take as long as they wanted for any decision. The players were instructed to not use any software to assist them with forming their strategy while playing.

The two human participants in the 1H+5AI experiment were Chris "Jesus" Ferguson and Darren Elias. Chris Ferguson has won more than $9.2 million playing live poker and has won six World Series of Poker events, including most famously the 2000 World Series of Poker Main Event. Darren Elias has won more than $7.1 million playing live poker and $3.8 million playing online poker. He holds four World Poker Tour titles and two World Championship of Online Poker titles. He is regarded as an elite player in this specific form of poker.

Both players were allowed to play at home on their own schedule. They could take as much time for any decision as they wanted and were told Pluribus would not change its strategy based on how long they took to make a decision. They were allowed—but not required—to play up to four tables simultaneously. Elias usually chose to play four tables while Ferguson usually

chose to play one.

The human participants in the 5H+1AI experiment were Jimmy Chou, Seth Davies, Michael Gagliano, Anthony Gregg, Dong Kim, Jason Les, Linus Loeliger, Daniel McAulay, Greg Merson, Nicholas Petrangelo, Sean Ruane, Trevor Savage, and Jacob Toole. All participants have won at least $1 million playing poker professionally.

Each human player was assigned an alias that they would use throughout the experiment. The humans could see the aliases of the humans they were playing with, but did not know the identity of the human behind the alias. We made no effort to mask the identity of Pluribus. On each day, five players were selected among those who volunteered to play from the pool of 13 players. In some cases due to player availability, the day was divided into multiple sessions with a different set of players in each session. The length of play on each day lasted between three hours and eight hours, but was typically four hours. There were breaks every two hours lasting at least ten minutes. The assignment of players to seats (i.e., player order) was determined randomly at the beginning of each day.

Players were asked, but not required, to play four tables simultaneously. This was later changed to a maximum of six tables at the request of the players (due to the lack of a time limit on decisions). The players played at a pace of about 180 hands per hour when playing four tables. The humans were asked to not try to uncover the identity of any other participant in the experiment, to not mention the experiment to anyone while it was happening, and to not intentionally collude with any player (which is not allowed in poker).

The players were paid a minimum of $0.40 per hand played, but this could increase to as much as $1.60 per hand based on performance. The compensation structure was designed to as closely as possible align the incentives of the players with those of a normal cash game, while still guaranteeing the players would not lose money and would divide a fixed-sized prize pool. Specifically, each player was paid $\$(1 + 0.005X)$ per hand, where X is the player's adjusted

variance-reduced win rate in terms of mbb/game with a floor at $X = -120$ and a ceiling at $X = 120$. Any player that lost more than -120 mbb/game or won more than 120 mbb/game after the application of variance reduction had the excess winnings or losses divided among all the players proportional to the number of hands each player played (after first canceling out all players' excess winnings with all players' excess losses as much as possible and canceling out all of Pluribus's winnings or losses). Players were not informed of their variance-reduced win rates until after the experiment was completed.

Statistical analysis

For the 1H+5AI experiments, each participant played 5,000 hands for a total of 10,000 hands. For the 5H+1AI experiment, a total of 10,000 hands were played. In both the 5H+1AI experiment and the 1H+5AI experiment, we used a one-tailed test to determine at the 95% confidence level whether Pluribus was stronger than the humans.

Each hand in each experiment format was treated as an independent and identically distributed sample. (Theoretically speaking, this assumption may not be exactly accurate because humans may adjust their strategy over the course of the experiment. Nevertheless, this assumption is the standard assumption that is used in this field of research.)

In the case of the 1H+5AI experiment, we only measure whether Pluribus is profitable against both players in aggregate since that was what the experiment was designed to measure. In fact, the only reason we used two humans is that it would have taken prohibitively long for one human to complete all of the 10,000 hands. That said, we also do report p-values for the individual players for completeness. Chris Ferguson was behind by an average of 25.2 mbb/game with a standard error of 20.2 mbb/game, which would be a p-value of 0.106. Darren Elias was behind by an average of 40.2 mbb/game with a standard error of 21.9, which would be a p-value of 0.033. Across both players, the average was 32.7 mbb/game with a standard error

of 14.9 mbb/game. Pluribus was determined to be profitable with a p-value of 0.014, which is statistically significant at the 95% confidence level, the standard threshold used in this field of research.

For the 5H+1AI experiment, Pluribus earned an average of 47.7 mbb/game with a standard error of 25.0 mbb/game. Pluribus was determined to be profitable with a p-value of 0.028, which is statistically significant at the 95% confidence level.

In the 5H+1AI experiment the human participants played an additional 13 hands beyond the requested 10,000. We did not include those additional hands in our analysis. With the additional hands included, Pluribus's win rate would increase to 50.9 mbb/game.

AIVAT could not be meaningfully applied for individual human players because AIVAT can only account for chance nodes, Pluribus's strategy across all possible hands, and Pluribus's probability distribution over actions. For human players, their probability distribution over actions and their strategy across all possible hands is unknown. Since most hands a human plays only minimally involves Pluribus, that only leaves chance nodes as something AIVAT can always control for. Table S1 shows results for each human participant (listing only their alias) after applying a relatively modest variance-reduction technique that for each player subtracts the expected value (according to Pluribus's blueprint) of each hand given all players' private cards.

Notation and background

In an imperfect-information extensive-form (i.e., tree-form) game there is a finite set of players, $\mathcal{P}$. A *node* (i.e., history) h is defined by all information of the current situation, including private knowledge known to only one player. $A(h)$ denotes the actions available at a node and $P(h)$ is either chance or the player whose turn it is to act at that node. We write $h \prec h'$ if a sequence of actions leads from h to h'. We represent the node that follows h after choosing action a by

$h \cdot a$. H is the set of all nodes. $Z \subseteq H$ are terminal nodes for which no actions are available and which award a value to each player.

Imperfect information is represented by *information sets* (infosets) for each player $p \in \mathcal{P}$. For any infoset I belonging to p, all nodes $h, h' \in I$ are indistinguishable to p. Moreover, every non-terminal node $h \in H$ belongs to exactly one infoset for each p.

A *strategy* (i.e., a *policy*) $\sigma(I)$ is a probability vector over actions for acting player p in infoset I. Since all states in an infoset belonging to p are indistinguishable, the strategies in each of them must be identical. The set of actions in I is denoted by $A(I)$. The probability of a particular action a is denoted by $\sigma(I, a)$ or by $\sigma(h, a)$. We define σ_p to be a strategy for p in every infoset in the game where p acts. A *strategy profile* σ is a tuple of strategies, one for each player.

We denote *reach* by $\pi^\sigma(h)$. This is the probability with which h is reached if all players play according to σ. $\pi^\sigma_p(h)$ is the contribution of p to this probability. $\pi^\sigma_{-p}(h)$ is the contribution of chance and all players other than p.

A *public state* G is defined as a set of nodes that an outside observer with no access to hidden information cannot distinguish. Formally, for any node $h \in G$, if $h, h' \in I$ for some infoset I, then $h' \in G$. An *imperfect-information subgame*, which we simply call a *subgame*, $\mathcal{S}$ is a union of public states where if any node A leads to any node B and both A and B are in $\mathcal{S}$, then every node between A and B is also in $\mathcal{S}$. Formally, for any node $h \in \mathcal{S}$, if $h, h' \in I$ for some infoset I then $h' \in \mathcal{S}$. Moreover, if $h \in \mathcal{S}$ and $h'' \in \mathcal{S}$ and there is a sequence of actions leading from h to h'' (i.e., $h \prec h''$), then for every node h' such that $h \prec h' \prec h''$, $h' \in \mathcal{S}$.

Variance reduction via AIVAT

We scored the players using a form of AIVAT modified to handle games with more than two players (*44*). AIVAT provides an unbiased estimate of the true win rate while reducing variance

in practice by about a factor of 9. We now provide a brief description of AIVAT.

Every time Pluribus acts at node h, its strategy defines a probability distribution over actions. Let $\sigma(h, a)$ be the probability that Pluribus chooses action a. Given all players' strategies, there is some unknown value $v(h, a)$ for choosing action a. Define $v(h) = \sum_{a \in A(h)} \sigma(h, a) v(h, a)$. $v(h)$ gets passed up to the parent node, which repeats until eventually we can compute $v(\emptyset)$ (the value for the root node, that is, the value for the entire hand). If we knew $v(h, a)$ exactly for each action a, then we could simply return $\sum_{a \in A(h)} \sigma(h, a) v(h, a)$. Unfortunately we do not know $v(h, a)$ because that would require knowledge of the other players' strategies. Instead, by choosing action a and playing the rest of the hand, Pluribus eventually receives some value $\hat{v}(h, a)$ that is a sample drawn from a distribution whose expectation is $v(h, a)$.

If we received one sample for each action, then we could return $\sum_{a \in A(h)} \sigma(h, a) \hat{v}(h, a)$ and in expectation this would be the same as $\sum_{a \in A(h)} \sigma(h, a) v(h, a)$. However, since only one action can be chosen during a hand, we define $\hat{v}'(h, a) = \hat{v}(h, a)$ if a was the one action that was sampled, and $\hat{v}'(h, a) = 0$ otherwise. Then $\sum_{a \in A(h)} \hat{v}'(h, a)$ is an unbiased estimate of $\sum_{a \in A(h)} \sigma(h, a) v(h, a)$ that only requires a sample for one action (though the variance may be quite high).

While we do not know $v(h, a)$ exactly, we may have a reasonable estimate of it. We can leverage this estimate to compute a lower-variance estimate of $\sum_{a \in A(h)} \sigma(h, a) v(h, a)$ that is still unbiased. Specifically, suppose we have arbitrary values $\tilde{v}(h, a)$ for each action a (where we would like $\tilde{v}(h, a) \approx v(h, a)$, but this is not required). Then $\sum_{a \in A(h)} \sigma(h, a) v(h, a) = \sum_{a \in A(h)} \sigma(h, a) \big(v(h, a) - \tilde{v}(h, a)\big) + \sum_{a \in A(h)} \sigma(h, a) \tilde{v}(h, a)$. Let a^* be the action that is randomly sampled according to probability distribution $\sigma(h)$. Then $\tilde{v}(h, a^*)$ is an unbiased estimate of $\sum_{a \in A(h)} \sigma(h, a) \tilde{v}(h, a)$ and, as previously established, $\hat{v}(h, a^*)$ is an unbiased estimate of $\sum_{a \in A(h)} \sigma(h, a) v(h, a)$. So $\hat{v}(h, a^*) - \tilde{v}(h, a^*) + \sum_{a \in A(h)} \sigma(h, a) \tilde{v}(h, a)$ is an unbiased estimate of $\sum_{a \in A(h)} \sigma(h, a) v(h, a)$. But if $\tilde{v}(h, a^*) \approx v(h, a^*)$, then this estimate will likely

have lower variance than $\hat{v}(h, a^*)$. Moreover, we can pass up the value $\hat{v}(h, a^*) - \tilde{v}(h, a^*) + \sum_{a \in A(h)} \sigma(h, a)\tilde{v}(h, a)$ rather than passing up $\hat{v}(h, a^*)$.

This variance-reduction approach also applies to chance nodes, because the probability distribution at chance nodes is known and fixed. However, it does not apply to human decision points, because the humans do not give a probability distribution for their actions.

Furthermore, Pluribus determines its strategy for every hand it could be holding (its entire range) rather than just the hand it was dealt. It can therefore estimate what its payoff would have been for each of these hands. This lowers variance even further.

Each copy of Pluribus runs AIVAT independently. The human's average win rate is simply the negative of the average of each AI's win rate. To reduce variance even further in the 1H+5AI experiments, we replay each hand with a copy of Pluribus in the human's position. We will refer to this copy of Pluribus in the human's position as the *Control*. The human's win rate is subtracted by the Control's win rate (which in expectation must be zero, since it is the same agent as its opponents). Moreover, if a hand ends with both the human and the Control folding in the first betting round, and the sequence of actions in the hand up to that point is identical in both cases, then that hand is treated as having zero expected value. This does not bias the result because the human and the Control do not make any further decisions in the hand, and have acted identically up to that point.

Hardware usage

Pluribus was trained on the Bridges supercomputer at the Pittsburgh Supercomputing Center. The blueprint strategy was trained on a single 64-core large shared memory node that has four 16-core Intel Xeon E5-8860 v3 CPUs. Although the node has 3 TB of memory available, less than 0.5 TB was needed. When playing in real time, Pluribus runs on a single 28-core, 128 GB regular shared memory node. The node uses two 14-core Intel Haswell E5-2695 v3 CPUs. No

GPUs were used at any point.

Further details of the abstraction algorithm

NLTH is a massive game that would be infeasible to store in memory. Abstraction reduces the size of NLTH by bucketing similar decision points together and by eliminating some actions from the game. A strategy is computed for this simplified, abstract version of the game and that strategy is used as a guide for how the full version of NLTH should be played. We used abstraction in Pluribus both for the precomputed blueprint strategy and for the strategies computed during real-time search.

There are two main forms of abstraction: action abstraction and information abstraction. Two infosets are bucketed together and treated identically if they share the same action-abstraction sequence and the same information-abstraction bucket.

Action abstraction involves reducing the number of actions that can be taken in the game. NLTH normally allows a player to raise any dollar increment between the minimum legal raise (which may be as low as \$100) and the number of remaining chips (which may be as high as \$10,000). The action abstraction we use allows between one and 14 different raise sizes depending on the particular decision point. All raise sizes are fractions of the size of the pot. These candidate raise sizes for Pluribus's algorithms to consider were decided by hand based on what raise sizes earlier versions of Pluribus decided to use with significant positive probabilities.

The action abstraction of the blueprint strategy is particularly fine-grained on the first betting round because Pluribus usually does not use real-time search on this round. The action abstraction is more coarse on the second betting round. On the third and fourth betting round, there are at most three raise sizes for the first raise in the round (either $0.5\times$ the size of the pot, $1\times$ the size of the pot, or all remaining chips), and at most two for the remaining raises in a round ($1\times$ the size of the pot or all remaining chips). During real-time search, the number of

raise sizes varies between one and six. The fold and call actions were always included when they were legal actions in both the blueprint and real-time search.

There are a total of 664,845,654 action sequences in the blueprint action abstraction. However, only 413,507,309 of them were ever encountered during training. To reduce memory usage, memory for the regrets of an action sequence was only allocated when the action sequence was encountered for the first time. During real-time search, the number of action sequences (considering only players' actions, not chance's actions) in the subgame action abstraction is between 100 and 2,000.

Information abstraction buckets together situations based on the revealed chance outcomes. In poker, this is the private cards dealt to the player and the public board cards. We refer to a sequence of revealed private cards and revealed board cards as an *information situation*. *Lossless* abstraction only buckets together strategically identical information situations (*46, 47*). For example, 2♠6♠ is strategically identical to 2♡6♡ on the first betting round, so there is no loss in strategy quality if these two hands are treated identically. If lossless abstraction were applied on each round, there would be 169, 1,286,792, 55,190,538, and 2,428,287,420 strategically unique information situations on the first, second, third, and fourth betting rounds, respectively. However, we only use lossless abstraction on the first betting round. *Lossy* abstraction buckets together information situations that are not strategically identical, but that ideally are strategically similar. We use lossy abstraction on each betting round after the first one, with 200 buckets per round. That is, on each round after the first one, each information situation is put into one of 200 buckets, with the information situations in each bucket being treated identically. Information situations were bucketed using k-means clustering on domain-specific features (*26*).

The real-time search algorithm only uses lossless information abstraction for the round it is in. For later rounds, it uses lossy information abstraction with 500 buckets per round. That is, on each round, each information situation is put into one of 500 buckets, with information

situations in the same bucket being treated identically. These buckets were determined separately for each "flop" (the first three revealed board cards) using an algorithm that considers the future potential of each poker hand (*28*). That algorithm combines the idea of potential-aware abstraction (*48*) with clustering based on earth-mover distance (*27*).

Further details of the blueprint computation algorithm

We computed the blueprint strategy for Pluribus using external-sampling Monte Carlo Counterfactual Regret Minimization (*29, 35*) with two important improvements.

First, we used linear weighting for both the regret and average strategies for the first 400 minutes of the run (*38*). (We stop the discounting after that because the time cost of doing the multiplications with the discount factor is not worth the benefit later on.) Specifically, after every 10 minutes for the first 400 minutes, the regrets and average strategies were discounted by $\frac{T/10}{T/10+1}$, where T is the number of minute that have elapsed since the start of training. Experiments in two-player NLTH subgames show that this modification speeds up convergence by about a factor of 3 (*38*).

Second, our MCCFR algorithm did not explore every traverser action on every iteration after the first 200 minutes. Instead, for 95% of iterations, traverser actions with regret below -300,000,000 were not explored unless those actions were on the final betting round or those actions immediately led to a terminal node. In the remaining 5% of iterations, every traverser action was explored. This "pruning" of negative-regret actions means iterations can be completed more quickly. More importantly, it also effectively leads to a finer-grained information abstraction. For example, on the second betting round there are on average 6,434 infosets per abstract infoset bucket. The strategy for that abstract infoset bucket must generalize across all of those 6,434 infosets. But with pruning, many of those 6,434 infosets are traversed only 5% as often, so the strategy for the abstract infoset can better focus on generalizing across infosets that

are likely to actually be encountered during strong play. Negative-regret pruning has previously been used with great success in two-player NLTH (*6, 33, 34*). In six-player NLTH, the benefit of this pruning is even larger in practice because good strategies in six-player NLTH involve folding most hands early in the game, so an even smaller fraction of the game tree is reached with positive probability by the final strategy. The pseudocode for the blueprint computation is shown in Algorithm 1.

To save memory, regrets were stored using 4-byte integers rather than 8-byte doubles. There was also a floor on regret at -310,000,000 for every action. This made it easier to unprune actions that were initially pruned but later improved. This also prevented integer overflows.

Since CFR's average strategy is not guaranteed to converge to a Nash equilibrium in six-player poker, there is no theoretical benefit to using the average strategy as opposed to the current strategy. Nevertheless, after the initial 800 minutes of training, we stored the average strategy for the first betting round and used this as the blueprint strategy for the first betting round. For situations after the first betting round, a snapshot of the current strategy was taken every 200 minutes after the initial 800 minutes of training. The blueprint strategy after the first betting round was constructed by averaging together these snapshots (*32*). Averaging together snapshots that were saved to disk rather than maintaining the average strategy in memory for situations after the first round reduced memory usage by nearly half, and also reduced the computational cost of each MCCFR iteration.

Further details of the real-time search algorithm

The real-time search component of Pluribus, which determines an improved strategy during actual play, is the most intricate component of the system. On the first betting round, real-time search is used if an opponent chooses a raise size that is more than $100 off from any raise size in the blueprint action abstraction and there are no more than four players remaining in the

Algorithm 1 MCCFR with Negative-Regret Pruning

The final strategy for infoset I is $\phi(I)$ normalized.

function MCCFR-P(T) ▷ Conduct External-Sampling Monte Carlo CFR with Pruning
  for $P_i \in \mathcal{P}$ **do**
    for $I_i \in \mathcal{I}_i$ where $P(I_i) = i$ **do**
      for $a \in A(I_i)$ **do**
        $R(I_i, a) \leftarrow 0$
        if $betting\_round(I_i) = 0$ **then**
          $\phi(I_i, a) \leftarrow 0$
  for $t = 1$ to T **do**
    for $P_i \in \mathcal{P}$ **do**
      if $t \mod Strategy\_Interval = 0$ **then** ▷ $Strategy\_Interval$ = 10,000 in Pluribus
        UPDATE-STRATEGY($\emptyset, P_i$)
      if $t > Prune\_Threshold$ **then** ▷ $Prune\_Threshold$ is 200 minutes in Pluribus
        $q \sim [0, 1)$ ▷ Sample from a uniform random distribution between 0 and 1
        if $q < 0.05$ **then**
          TRAVERSE-MCCFR($\emptyset, P_i$)
        else
          TRAVERSE-MCCFR-P($\emptyset, P_i$)
      else
        TRAVERSE-MCCFR($\emptyset, P_i$)
    if $t < LCFR\_Treshold$ and $t \mod Discount\_Interval = 0$ **then**
      $d \leftarrow \frac{t/Discount\_Interval}{t/Discount\_Interval+1}$ ▷ $Discount\_Interval$ is 10 minutes in Pluribus
      for $P_i \in \mathcal{P}$ **do**
        for $I_i \in \mathcal{I}_i$ where $P(I_i) = i$ **do**
          for $a \in A(I_i)$ **do**
            $R(I_i, a) \leftarrow R(I_i, a) \cdot d$
            $\phi(I_i, a) \leftarrow \phi(I_i, a) \cdot d$
  return ϕ

function CALCULATE-STRATEGY($R(I_i), I_i$) ▷ Calculates the strategy based on regrets
  sum $\leftarrow 0$
  for $a \in A(I_i)$ **do**
    sum $\leftarrow$ *sum* $+ R_+(I_i, a)$
  for $a \in A(I_i)$ **do**
    if *sum* > 0 **then**
      $\sigma(I_i, a) \leftarrow \frac{R_+(I_i,a)}{sum}$
    else
      $\sigma(I_i, a) \leftarrow \frac{1}{|A(I_i)|}$
  return $\sigma(I_i)$

function UPDATE-STRATEGY(h, P_i) ▷ Update the average strategy for P_i
  if h is terminal or P_i not in hand or $betting\_round(h) > 0$ **then**
    return ▷ Average strategy only tracked on first betting round
  else if h is a chance node **then**
    $a \sim \sigma(h)$ ▷ Sample an action from the chance probabilities
    UPDATE-STRATEGY($h \cdot a, P_i$)
  else if $P(h) = P_i$ **then**
    $I_i \leftarrow I_i(h)$ ▷ The P_i infoset of this node
    $\sigma(I_i) \leftarrow$ CALCULATE-STRATEGY($R(I_i), I_i$) ▷ Determine the strategy at this infoset
    $a \sim \sigma(I_i)$ ▷ Sample an action from the probability distribution
    $\phi(I_i, a) \leftarrow \phi(I_i, a) + 1$ ▷ Increment the action counter
    UPDATE-STRATEGY($h \cdot a, P_i$)
  else
    for $a \in A(h)$ **do**
      UPDATE-STRATEGY($h \cdot a, P_i$) ▷ Traverse each action

function TRAVERSE-MCCFR(h, P_i) ▷ Update the regrets for P_i
  if h is terminal **then**
    return $u_i(h)$
  else if P_i not in hand **then**
    return TRAVERSE-MCCFR($h \cdot 0, P_i$) ▷ The remaining actions are irrelevant to P_i
  else if h is a chance node **then**
    $a \sim \sigma(h)$ ▷ Sample an action from the chance probabilities
    return TRAVERSE-MCCFR($h \cdot a, P_i$)
  else if $P(h) = P_i$ **then**
    $I_i \leftarrow I_i(h)$ ▷ The P_i infoset of this node
    $\sigma(I_i) \leftarrow$ CALCULATE-STRATEGY($R(I_i), I_i$) ▷ Determine the strategy at this infoset
    $v(h) \leftarrow 0$ ▷ Initialize expected value at zero
    for $a \in A(h)$ **do**
      $v(h, a) \leftarrow$ TRAVERSE-MCCFR($h \cdot a, P_i$) ▷ Traverse each action
      $v(h) \leftarrow v(h) + \sigma(I_i, a) \cdot v(h, a)$ ▷ Update the expected value
    for $a \in A(h)$ **do**
      $R(I_i, a) \leftarrow R(I_i, a) + v(h, a) - v(h)$ ▷ Update the regret of each action
    return $v(h)$ ▷ Return the expected value
  else
    $I_{P(h)} \leftarrow I_{P(h)}(h)$ ▷ The $P_{P(h)}$ infoset of this node
    $\sigma(I_{P(h)}) \leftarrow$ CALCULATE-STRATEGY($R(I_{P(h)}), I_{P(h)}$)
    $a \sim \sigma(I_{P(h)})$ ▷ Sample an action from the probability distribution
    return TRAVERSE-MCCFR($h \cdot a, P_i$)

```
function TRAVERSE-MCCFR-P(h, P_i)                    ▷ MCCFR with pruning for very negative regrets
    if h is terminal then
        return u_i(h)
    else if P_i not in hand then
        return TRAVERSE-MCCFR-P(h · 0, P_i)          ▷ The remaining actions are irrelevant to P_i
    else if h is a chance node then
        a ∼ σ(h)                                     ▷ Sample an action from the chance probabilities
        return TRAVERSE-MCCFR-P(h · a, P_i)
    else if P(h) = P_i then
        I_i ← I_i(h)                                 ▷ The P_i infoset of this node
        σ(I_i) ← CALCULATE-STRATEGY(R(I_i), I_i)     ▷ Determine the strategy at this infoset
        v(h) ← 0                                     ▷ Initialize expected value at zero
        for a ∈ A(h) do
            if R(I_i, a) > C then                    ▷ C is -300,000,000 in Pluribus
                v(h, a) ← TRAVERSE-MCCFR-P(h · a, P_i)
                explored(a) ← True
                v(h) ← v(h) + σ(I_i, a) · v(h, a)    ▷ Update the expected value
            else
                explored(a) ← False
        for a ∈ A(h) do
            if explored(a) = True then
                R(I_i, a) ← R(I_i, a) + v(h, a) − v(h)   ▷ Update the regret for this action
        return v(h)                                  ▷ Return the expected value
    else
        I_P(h) ← I_P(h)(h)                           ▷ The P_P(h) infoset of this node
        σ(I_P(h)) ← CALCULATE-STRATEGY(R(I_P(h)), I_P(h))
        a ∼ σ(I_P(h))                                ▷ Sample an action from the probability distribution
        return TRAVERSE-MCCFR-P(h · a, P_i)
```

hand. Otherwise, Pluribus uses the randomized pseudo-harmonic *action translation* algorithm (which empirically has the lowest exploitability of all known action translation algorithms) to map the raise to a nearby size and proceeds to play according to the blueprint strategy as if that mapped raise size had been chosen (*39*). Real-time search is always used to determine the strategy on the second, third, and fourth betting rounds.

Structure of imperfect-information subgames in Pluribus

In perfect-information games, search begins at a *root node*, which is the current state of the world. Players are able to change their strategy below the root node until a *leaf node* of the search space is reached (or a terminal node that ends the game is reached). A leaf node's value is fixed and ideally approximates the value that would ensue if all players played perfectly beyond the leaf node. In practice, this value can be estimated by, for example, using a game-specific heuristic or by assuming that all players play according to the blueprint strategy after the leaf node. The root, the leaves, and the nodes in between them constitute a *subgame*. For perfect-information games, if the value of each leaf node equals the value of both players playing perfectly from that point forward and the subgame is solved exactly (i.e., no player can do better given every other player's strategy in the subgame), then the solution to the subgame is part of a Nash equilibrium strategy. Thus, search is a method of achieving in real time a closer approximation of a Nash equilibrium compared to the blueprint strategy.

Pluribus uses a generalized form of this search paradigm that allows it to be run in imperfect-information games. The two main modifications are how the root of the subgame is represented and how the values at the leaf nodes are calculated.

Since players do not know the exact node they are in when playing an imperfect-information game, it is not possible to have a single root node. Instead, the "root" of a subgame in Pluribus is a probability distribution over the nodes in a public state G. The probability of a node is the

normalized probability that that node would be reached assuming that all players play according to a strategy profile σ. Formally, the root of an imperfect-information subgame in Pluribus is a chance node with outcomes that lead to each node in the root public state. The probability of the chance node outcome leading to node h is $\frac{\pi^\sigma(h)}{\sum_{h' \in G} \pi^\sigma(h')}$, where G is the root public state.

If the root probability distribution is correct (that is, all players were indeed playing according to σ), then solving the remainder of the game starting at the root public state would produce an optimal strategy going forward (if there are only two players remaining in the hand, or if all remaining players play the same solution). Of course, it is not possible for Pluribus to know the strategy profile σ that all players played (and therefore know the true probability distribution over nodes in G) because Pluribus does not have access to the other players' strategies, only their observed actions.

To handle this problem, Pluribus calculates what its strategy would be in each situation where an opponent has acted, and assumes the opponent followed that strategy. We refer to this as *unsafe* search (*49*). Unsafe search lacks theoretical guarantees on performance even in two-player zero-sum games and there are cases where it leads to highly exploitable strategies. While *safe search* alternatives exist that have provable guarantees on exploitability in two-player zero-sum games (*50–52*), in practice they tend to do worse than modern, careful unsafe search in head-to-head performance. Unsafe search has the added benefit that it is not necessary to compute a strategy for a hand that has zero probability. In six-player poker, most hands are folded with 100% probability in the first action, so whenever search is conducted, there is typically only a small fraction of hands for which a strategy actually needs to be computed. This makes unsafe search faster by about a factor of four.

To obtain the practical benefits of unsafe search while mitigating the potential for highly exploitable strategies, we use a new form of nested search in this paper that is similar to nested unsafe search but which always solves starting at the beginning of the current betting round

(described in the body of this paper) rather than starting at the most recent decision point. The traversers strategy is held fixed for actions the traverser has already chosen in the betting round. Experimental results show that if the root public state follows a chance node with a large branching factor, as is always done in this form of nested search, then unsafe search typically produces strategies with low exploitability in two-player zero-sum games (*52*). This mitigates the potential for unsafe search to be exploitable, while still retaining the practical average benefits of unsafe search.

To implement this form of search, Pluribus maintains a probability distribution over pairs of private cards a player (including Pluribus itself) could be holding from an outside observer's perspective based on the actions that have occurred so far in the game. Since there are $\binom{52}{2} = 1326$ combinations of private cards that a player can have, each probability is initially $\frac{1}{1326}$. Whenever a round ends, Pluribus makes the public state at the start of the new round the root of the new subgame. It also updates its belief distribution for each player's possible private cards using Bayes' rule based on strategy profile σ, where σ is the blueprint if real-time search has not yet been conducted; otherwise σ is the output of the previously-run search.

If search is being done on the first betting round, then the subgame extends to the end of the round, with leaf nodes at the chance nodes at the start of the second round. If search is being done on the second betting round and there were more than two players at the start of the round, then leaf nodes occur either at the chance nodes at the stat of the third betting round or immediately after the second raise action, whichever is earlier. In all other cases, the subgame extends to the end of the game.

Pluribus used one of two different forms of CFR to compute a strategy in the subgame depending on the size of the subgame and the part of the game. If the subgame is relatively large or it is early in the game, then Monte Carlo Linear CFR is used just as it was for the blueprint strategy computation. Otherwise, Pluribus uses an optimized vector-based form of

Linear CFR (*38*) that samples one set of public board cards per thread (*42*). In both cases, Pluribus actually plays according to the strategy on the final iteration rather than the weighted average strategy over all iterations. However, σ is updated based on the weighted average strategy. Playing according to the final iteration helps Pluribus avoid poor actions that are not completely eliminated in CFR's weighted average strategy. This could potentially come at the cost of increased exploitability, but in practice the final iteration's strategy is sufficiently unpredictable that any exploitation is infeasible.

Abstraction and off-tree actions in subgames

When conducting search, the current betting round (the round of the root public state) uses lossless information abstraction, but subsequent betting rounds use lossy information abstraction in which each information situation is assigned to one of 500 buckets per round. Each information situation in a bucket is treated identically. Only a small number of possible raise actions are included in the subgame action abstraction, typically no more than five. If an opponent chooses an action that was not included in the action abstraction, then that action is added as a valid action into the subgame model and the subgame is searched again from the root (which is typically the start of the betting round). The root does not change until a new round is reached.

However, there is a risk that the strategy resulting from the second search might be different than the strategy that Pluribus has played so far with its actual hand. For example, there could be actions A and B at the root that are distinct actions but nevertheless the game trees following each are identical, so there is effectively no difference between them. The first time the subgame is searched, the solution might call for Pluribus to always choose A at the root and therefore Pluribus would choose A. However, the second time the subgame is searched, the solution might call for Pluribus to always choose B at the root even though Pluribus had already chosen A. Since the new solution assumes Pluribus would always choose B at the root and never A,

the new strategy beyond A might be nonsensical. Not accounting for this problem could cause Pluribus to choose a poor strategy when search is conducted in later situations.

One way to prevent this would be to freeze the strategy for decision points preceding the current public state that Pluribus is in. That would effectively always make the current public state the root of the subgame. However, freezing the strategy for opponent decision points would make Pluribus less robust to the possibility of an opponent shifting to a different strategy.

Instead, we only freeze the action probabilities for any actions Pluribus chose so far in the subgame. The action probabilities are only frozen for its actual hand, not for its other possible hands. Opponent action probabilities are also not frozen. However, when a betting round ends, the root of the subgame is changed to the start of the new betting round, which effectively freezes the strategy of all infosets preceding the new root public state.

Leaf node values in imperfect-information subgames

Assuming that all players play according to the blueprint strategy profile following leaf nodes, as is often done in perfect-information games, can lead to highly exploitable strategies when doing search in imperfect-information subgames even if the blueprint is an exact Nash equilibrium because the opponents could shift to different strategies that may exploit the searcher.

To address this in Pluribus, when a leaf node is reached in a subgame during search, each player still in the hand simultaneously chooses one of four different continuation strategies to play for the remainder of the game. They may also choose any mixture of (i.e., probability distribution over) the four strategies. The choice a player makes must be identical for all leaf nodes that the player cannot distinguish among (i.e., it must be identical for all leaf nodes that are in the same infoset). This final choice of strategy for the remainder of the game is essentially just another action in the subgame and is selected via the search algorithm (in the

case of Pluribus, our improved MCCFR algorithm discussed earlier in this paper).[1]

In different applications, the set of possible continuations strategies could be chosen in various different ways, and there could be more or less than four of them per player. In Pluribus, the four possible continuation strategies are biased modifications of the blueprint strategy. The first is simply the unaltered blueprint. The second is the blueprint strategy biased toward folding. Specifically, the probability of folding in all decision points is multiplied by 5 and the probabilities are then renormalized. The third is the blueprint strategy biased toward calling by multiplying the call probability by 5 and then renormalizing all the probabilities. The fourth is the blueprint strategy biased toward raising by multiplying all raise action probabilities by 5 and then renormalizing all the probabilities. To reduce memory usage, the continuation strategies are compressed by sampling an action at each abstract infoset according to the probability distribution at that abstract infoset. Rather than storing the regrets or probabilities for each action, only that action is stored (using the minimum number of bits necessary). This is unlikely to bias the results because the odds of encountering the same abstract infoset multiple times is very low.

If an opponent chose an action earlier in the game that was not in the blueprint action abstraction, then a leaf node in the subgame may not exist in the blueprint abstraction. In that case we map the leaf node to the nearest node in the blueprint action abstraction using the deterministic pseudo-harmonic action translation (*39*).

[1]We used a similar technique in the two-player NLTH poker AI Modicum (*41*), but in that case only the opponent chose among the continuation strategies while the traversing player could only choose the blueprint strategy. That is sound in theory for two-player zero-sum games, but in practice gives more power to the opponent and therefore results in the traversing player playing a more conservative, defensive strategy that has lower expected value. Allowing the traversing player to also choose among continuation strategies helps balance the strength of the players and helps address this problem. This is also sound in theory for two-player zero-sum games.

Algorithm 2 Nested search used in Pluribus

$I \leftarrow \emptyset$ ▷ Initialize our current infoset I as the infoset at the start of the game
$G_{root} \leftarrow G(I)$ ▷ Initialize the subgame root as the public node of I
$\sigma \leftarrow \sigma_{blueprint}$ ▷ Initialize the game's strategy profile as the blueprint

function OPPONENTTURN(a) ▷ Opponent chose action a
    if $!InAbstraction(I, a)$ **then** ▷ If action a is not already in the action abstraction for infoset I
        for each node $h \in G(I)$ **do**
            $AddAction(h, a)$ ▷ Add a as a legal action for all nodes in the public node of I
        $\sigma \leftarrow Search(G_{root})$ ▷ Compute a new strategy profile for the subgame starting at the root
    $I \leftarrow I \cdot a$ ▷ Advance the current infoset from I to $I \cdot a$
    $CheckNewRound()$

function OURTURN ▷ It is our turn to act
    $a \sim \sigma(I)$ ▷ Sample an action from the probability distribution at this infoset
    $frozen(I) = True$ ▷ $\sigma(I)$ won't change if a new strategy is computed for the subgame
    $I \leftarrow I \cdot a$ ▷ Advance the current infoset from I to $I \cdot a$
    $CheckNewRound()$

function CHECKNEWROUND ▷ Check if a new round is reached. If so, update the root.
    if $BettingRound(G(I)) > BettingRound(G_{root})$ **then**
        $G_{root} = G(I)$ ▷ Update the root public node
        $\sigma \leftarrow Search(G_{root})$ ▷ Compute a new strategy profile for the subgame starting at the root

Participant Alias	Hands Played	Participant Win Rate	Standard Error
Participant A	9,121	141.8 mbb/game	85.8 mbb/game
Participant B	7,512	-86.7 mbb/game	87.7 mbb/game
Participant C	6,713	-49.9 mbb/game	121.9 mbb/game
Participant D	6,055	4.9 mbb/game	122.2 mbb/game
Participant E	5,510	101.5 mbb/game	121.8 mbb/game
Participant F	4,483	-59.7 mbb/game	108.6 mbb/game
Participant G	2,560	-126.9 mbb/game	133.5 mbb/game
Participant H	2,509	229.7 mbb/game	177.6 mbb/game
Participant I	1,535	-131.7 mbb/game	229.9 mbb/game
Participant J	1,378	89.3 mbb/game	237.6 mbb/game
Participant K	1,365	141.0 mbb/game	284.1 mbb/game
Participant L	771	-35.6 mbb/game	418.6 mbb/game
Participant M	488	-492.5 mbb/game	515.4 mbb/game

Table S1: The number of hands played, win rate after modest variance reduction, and standard error after modest variance reduction for each human participant in the 5H+1AI experiment. Due to the extremely high variance in no-limit poker and the small sample size, no meaningful conclusions can be drawn about the performance of any individual participant. Only Pluribus's performance after the application of AIVAT can be meaningfully evaluated.