Science

RESEARCH ARTICLES

Cite as: N. Brown, T. Sandholm, *Science* 10.1126/science.aao1733 (2017).

Superhuman AI for heads-up no-limit poker: Libratus beats top professionals

Noam Brown and Tuomas Sandholm*

Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA.

*Corresponding author. Email: sandholm@cs.cmu.edu

No-limit Texas hold'em is the most popular form of poker. Despite AI successes in perfect-information games, the private information and massive game tree have made no-limit poker difficult to tackle. We present Libratus, an AI that, in a 120,000-hand competition, defeated four top human specialist professionals in heads-up no-limit Texas hold'em, the leading benchmark and long-standing challenge problem in imperfect-information game solving. Our game-theoretic approach features application-independent techniques: an algorithm for computing a blueprint for the overall strategy, an algorithm that fleshes out the details of the strategy for subgames that are reached during play, and a self-improver algorithm that fixes potential weaknesses that opponents have identified in the blueprint strategy.

In recent years the field of artificial intelligence (AI) has advanced considerably. The measure of this progress has, in many cases, been marked by performance against humans in benchmark games. AI programs have defeated top humans in checkers (*1*), chess (*2*), and Go (*3*). In these perfect-information games both players know the exact state of the game at every point. In contrast, in imperfect-information games, some information about the state of the game is hidden from a player—for example, the opponent may hold hidden cards. Hidden information is ubiquitous in real-world strategic interactions, such as business strategy, negotiation, strategic pricing, finance, cybersecurity, and military applications, which makes research on general-purpose techniques for imperfect-information games particularly important.

Hidden information makes a game far more complex for a number of reasons. Rather than simply search for an optimal sequence of actions, an AI for imperfect-information games must determine how to balance actions appropriately, so that the opponent never finds out too much about the private information the AI has. For example, bluffing is a necessary feature in any competitive poker strategy, but bluffing all the time would be a bad strategy. In other words, the value of an action depends on the probability it is played.

Another key challenge is that different parts of the game cannot be considered in isolation; the optimal strategy for a given situation may depend on the strategy that would be played in situations that have not occurred (*4*). As a consequence, a competitive AI must always consider the strategy for the game as a whole.

Poker has a long history as a challenge problem for developing AIs that can address hidden information (*5*–*11*). No-limit Texas hold'em is the most popular form of poker in the world. The heads-up (that is, two-player) variant prevents opponent collusion and kingmaker scenarios where a bad player causes a mediocre player to shine, and therefore allows a clear winner to be determined. Due to its large size and strategic complexity, heads-up no-limit Texas hold'em (HUNL) has been the primary benchmark and challenge problem for imperfect-information game solving for several years. No prior AI has defeated top human players in this game.

In this paper we introduce Libratus, (*12*) an AI that takes a distinct approach to addressing imperfect-information games. In a 20-day, 120,000-hand competition featuring a $200,000 prize pool, it defeated top human professionals in HUNL. The techniques in Libratus do not use expert domain knowledge or human data and are not specific to poker; thus they apply to a host of imperfect-information games.

Game-solving approach in Libratus

Libratus features three main modules:

(i) The first module computes an abstraction of the game, which is smaller and easier to solve, and then computes game-theoretic strategies for the abstraction. The solution to this abstraction provides a detailed strategy for the early rounds of the game, but only an approximation for how to play in the more numerous later parts of the game. We refer to the solution of the abstraction as the blueprint strategy.

(ii) When a later part of the game is reached during play, the second module of Libratus constructs a finer-grained abstraction for that subgame and solves it in real time (*13*). Unlike subgame-solving techniques in perfect-information games, Libratus does not solve the subgame abstraction in isolation; instead, it ensures that the fine-grained solution to the subgame fits within the larger blueprint strategy of the whole game. The subgame solver has several key advantages over prior subgame-solving techniques (*14*, *15*, *16*). Whenever the opponent makes a move that is not in the abstraction, a subgame is solved with that action included. We call this nested subgame solving. This technique comes with a provable safety guarantee.

(iii) The third module of Libratus—the self-improver—enhances the blueprint strategy. It fills in missing branches in the blueprint abstraction and computes a game-theoretic strategy for those branches. In principle, one could conduct all such computations in advance, but the game tree is way too large for that to be feasible. To tame this complexity, Libratus uses the opponents' actual moves to suggest where in the game tree such filling is worthwhile.

In the following three subsections, we present these three modules in more detail.

Abstraction and equilibrium finding: Building a blueprint strategy

One solution to the problem of imperfect information is to simply reason about the entire game as a whole, rather than just pieces of it. In this approach, a solution is pre-computed for the entire game, possibly using a linear program (*10*) or an iterative algorithm (*17*–*21*). For example, an iterative algorithm called counterfactual regret minimization plus (CFR+) was used to near-optimally solve heads-up limit Texas hold'em, a relatively simple version of poker, which has about 10^{13} unique decision points (*11*, *22*).

In contrast, HUNL (*23*) has 10^{161} decision points (*24*), so traversing the entire game tree even once is impossible. Precomputing a strategy for every decision point is infeasible for such a large game.

Fortunately, many of those decision points are very similar. For example, there is little difference between a bet of $100 and a bet of $101. Rather than consider every possible bet between $100 and $20,000, we could instead just consider increments of $100. This is referred to as action abstraction. An abstraction is a smaller, simplified game that retains as much as possible the strategic aspects of the original game. This drastically reduces the complexity of solving the game. If an opponent bets $101 during an actual match, then the AI may simply round this to a bet of $100 and respond accordingly (*25*–*27*). Most of the bet sizes included in Libratus's action abstraction were nice fractions or multiples of the pot [roughly determined by analyzing the most common bet sizes at various points in the game taken by prior top AIs in the Annual Computer Poker Competition (ACPC) (*28*)]. However, certain bet sizes early in the game tree were determined by an application-independent parameter optimization algorithm that converged to a locally optimal set of bet sizes (*29*).

An additional form of abstraction is abstraction of actions taken by chance, that is, card abstraction in the case of poker. Similar hands are grouped together and treated identically. Intuitively, there is little difference between a King-high flush and a Queen-high flush. Treating those hands as identical reduces the complexity of the game and thus makes it computationally easier. Nevertheless, there are still differences even between a King-high flush and a Queen-high flush. At the highest levels of play, those distinctions may be the difference between winning and losing. Libratus does not use any card abstraction on the first and second betting rounds. The last two betting rounds, which have a significantly larger number of states, are abstracted only in the blueprint strategy. The 55 million different hand possibilities on the third round were algorithmically grouped into 2.5 million abstract buckets, and the 2.4 billion different possibilities on the fourth round were algorithmically grouped into 1.25 million abstract buckets. However, the AI does not follow the blueprint strategy in these rounds and instead applies nested subgame solving, described in the next section, which does not use any card abstraction. Thus, each poker hand is considered individually during actual play. The card abstraction algorithm that we used was similar to that used in our prior AIs Baby Tartanian8 (*30*), which won the 2016 ACPC, and Tartanian7 (*31*–*33*), which won the 2014 ACPC (there was no ACPC in 2015).

Once the abstraction was constructed, we computed the blueprint strategy for Libratus by having the AI play simulated games of poker against itself (while still exploring the hypothetical outcomes of actions not chosen) using an improved version of an algorithm called Monte Carlo Counterfactual Regret Minimization (MCCFR). MCCFR (*17*, *34*, *35*) has a long history of use in successful poker AIs (*30*, *31*, *36*, *37*). MCCFR maintains a regret value for each action. Intuitively, regret represents how much the AI regrets having not chosen that action in the past. When a decision point is encountered during self play, the AI chooses actions with higher regret with higher probability (*38*). As more and more games are simulated, MCCFR guarantees that with high probability a player's average regret for any action (total regret divided by the number of iterations played) approaches zero. Thus, the AI's average strategy over all simulated games gradually improves. We will now describe the equilibrium-finding algorithm (*4*).

On each simulated game, MCCFR chooses one player (who we refer to as the traverser) that will explore every possible action and update his regrets, while the opponent simply plays according to the strategy determined by the current regrets. The algorithm switches the roles of the two players after each game, that is, a single hand of poker. Every time either player is faced with a decision point in a simulated game, the player will choose a probability distribution over actions based on regrets on those actions (which are determined by what he had learned in earlier games when he had been in that situation). For the first game, the AI has not learned anything yet and therefore uses a uniform random distribution over actions. At traverser decision points, MCCFR explores every action in a depth-first manner. At opponent decision points, MCCFR samples an action based on the probability distribution. This process repeats at every decision point until the game is over and a reward is received, which is passed up. When a reward is returned by every action at a traverser decision point, MCCFR calculates the weighted average reward for that decision point based on the probability distribution over actions. The regret for each action is then updated by adding the value returned by that action, and subtracting the weighted average reward for the decision point. The weighted average reward is then passed up to the preceding decision point, and so on.

Our improved version of MCCFR traverses a smaller portion of the game tree on each iteration. Intuitively, there are many clearly suboptimal actions in the game, and repeatedly exploring them wastes computational resources that could be better used to improve the strategy elsewhere. Rather than explore every hypothetical alternative action to see what its reward would have been, our algorithm probabilistically skips over unpromising actions that have very negative regret as it proceeds deeper into the tree during a game (*30*, *39*). This led to a factor of three speedup of MCCFR in practice and allowed us to solve larger abstractions than were otherwise possible.

This skipping also mitigates the problems that stem from imperfect recall. The state-of-the-art practical abstractions in the field, including ours, are imperfect-recall abstractions where some aspects of the cards on the path of play so far are intentionally forgotten in order to be able to computationally afford to have a more detailed abstraction of the present state of cards (*30*–*32*, *40*). Since all decisions points in a single abstract card bucket share the same strategy, updating the strategy for one of them leads to updating the strategy for all of them. This is not an issue if all of them share the same optimal strategy at the solution reached, but in practice there are differences between their optimal strategies and they effectively "fight" to push the bucket's strategy toward their own optimal strategy. Skipping negative-regret actions means that decision points that will never be reached in actual play will no longer have their strategies updated, thereby allowing the decision points that will actually occur during play to move the bucket's strategy closer to their optimal strategies.

We ran our algorithm on an abstraction that is very detailed in the first two rounds of HUNL, but relatively coarse in the final two rounds. However, Libratus never plays according to the abstraction solution in the final two rounds. Rather, it uses the abstract blueprint strategy in those rounds only to estimate what reward a player should expect to receive with a particular hand in a subgame. This estimate is used to determine a more precise strategy during actual play, as described in the next section.

Nested safe subgame solving

Although purely abstraction-based approaches have produced strong AIs for poker (*25*, *30*, *32*, *41*), abstraction alone has not been enough to reach superhuman performance in HUNL. In addition to abstraction, Libratus builds upon prior research into subgame solving (*14*–*16*, *42*), in which a more detailed strategy is calculated for a particular part of the game that is reached during play. Libratus features many advances in subgame solving that proved critical to achieving superhuman performance (*43*).

Libratus plays according to the abstract blueprint strategy only in the early parts of HUNL, where the number of possible states is relatively small and we can afford the abstraction to be extremely detailed. Upon reaching the third betting round, or any earlier point in the game where the remaining game tree is sufficiently small (*44*), Libratus constructs a new, more detailed abstraction for the remaining subgame and solves it in real time.

However, there is a major challenge with subgame solving in imperfect-information games: a subgame cannot be solved in isolation because its optimal strategy may depend on other, unreached subgames (*4*). Prior AIs that used real-time subgame solving addressed this problem by assuming the opponent plays according to the blueprint strategy. However, the opponent can exploit this assumption by simply switching to a different strategy. For this reason, the technique may produce strategies that are far worse than the blueprint strategy and is referred to as unsafe subgame solving (*42*, *45*). Safe subgame solving techniques, on the other hand, guarantee that the subgame's new strategy makes the opponent no better off no matter what strategy the opponent might use (*14*). They accomplish this by ensuring that the new strategy for the subgame fits within the overarching blueprint strategy of the original abstraction. Ensuring the opponent is no better off relative to the blueprint strategy is trivially possible because we could just reuse the blueprint strategy. However, now that the abstraction is more detailed in the subgame and we can better distinguish the strategic nuances of the subgame, it may be possible to find an improvement over the prior strategy that makes the opponent worse off no matter what cards she is holding.

We now describe Libratus's core technique for determining an improved strategy in a subgame. For exposition, we assume Player 2 (P2) is determining an improved strategy against Player 1 (P1). Given that P2's strategy outside the subgame is σ_2, there exists some optimal strategy σ_2^* that P2 could play in the subgame. We would like to find or approximate σ_2^* in real time. We assume that, for each poker hand P1 might have, we have a good estimate of the value P1 receives in the subgame with that hand by playing optimally against σ_2^*, even though we do not know σ_2^* itself. Although we do not know these values exactly, we can approximate them with the values P1 receives in the subgame in the blueprint strategy. We later prove that if these estimates are approximately accurate, we can closely approximate σ_2^*.

To find a strategy close to σ_2^* in the subgame using only the values from the blueprint, we create an augmented subgame (Fig. 1) which contains the subgame and additional structures. At the start of the augmented subgame, P1 is privately dealt a random poker hand. Given that P2 plays according to σ_2 prior to the subgame, and given P1's dealt hand, there is a particular probability distribution over what hands P2 might have in this situation. P2 is dealt a poker hand according to this probability distribution. P1 then has the choice of entering the subgame (which is now far more detailed than in the blueprint strategy), or of taking an alternative payoff that ends the augmented subgame immediately. The value of the alternative payoff is our estimate, according to the blueprint strategy, of P1's value for that poker hand in that subgame. If P1 chooses to enter the subgame, then play proceeds normally until the end of the game is reached. We can solve this augmented subgame just as we did for the blueprint strategy (*46*).

For any hand P1 might have, P1 can do no worse in the augmented subgame than just choosing the alternative payoff (which awards our estimate of the expected value P1 could receive against σ_2^*). At the same time, P2 can ensure that for every poker hand P1 might have, he does no better than what he could receive against σ_2^*, because P2 can simply play σ_2^* itself. Thus, any solution to the augmented subgame must do approximately as well as σ_2^*—where the approximation error depends on how far off our estimates of P1's values are. P2 then uses the solution to the augmented subgame as P2's strategy going forward.

All of this relied on the assumption that we have accurate estimates of P1's values against σ_2^*. Although we do not know these values exactly, we can approximate them with values from the blueprint strategy. We now prove that if these estimates are approximately accurate, subgame solving will produce a strategy that is close to the quality of σ_2^*. Specifically, we define the exploitability of a strategy σ_2 as how much more σ_2 would lose, in expectation, against a worst-case opponent than what P2 would lose, in expectation, in an exact solution of the full game.

Theorem 1 uses a form of safe subgame solving we coin Estimated-Maxmargin. We define a margin for every P1 hand in a subgame as the expected value of that hand according to the blueprint minus what P1 could earn with that hand, in expectation, by entering the more-detailed subgame. Estimated-Maxmargin finds a strategy that maximizes the minimum margin among all P1 hands. It is similar to a prior technique called Maxmargin (*15*) except that the prior technique conservatively used as the margin what P1 could earn in the subgame, in expectation, by playing a best response to P2's blueprint strategy minus what P1 could earn, in expectation, by entering the more-detailed subgame.

Theorem 1. *Let σ_i be a strategy for a two-player zero-sum perfect-recall game, let S be a set of non-overlapping subgames in the game, and let σ_i^* be the least-exploitable strategy that differs from σ_i only in S. Assume that for any opponent decision point (hand in the case of poker) and any subgame in S, our estimate of the opponent's value in a best response to σ_i^* for that decision point in that subgame is off by at most Δ. Applying Estimated-Maxmargin subgame solving to any subgame in S reached during play results in overall exploitability at most 2Δ higher than that of σ_i^* (47).*

Although safe subgame solving techniques have been known for three years (*14*, *15*), they were not used in practice because empirically they performed significantly worse than unsafe subgame solving (*42*) head to head (*48*). Libratus features a number of advances to subgame solving that greatly improve effectiveness.

(i) Although we describe safe subgame solving as using estimates of P1 values, past techniques used upper bounds on those values (*14*, *15*). Using upper bounds guarantees that the subgame solution has exploitability no higher than the blueprint strategy. However, it tends to lead to overly conservative strategies in practice. Using estimates can, in theory, result in strategies with higher exploitability than the blueprint strategy, but Theorem 1 bounds how much higher this exploitability can be.

(ii) It arrives at better strategies in subgames than was previously thought possible. Past techniques ensured that the new strategy for the subgame made P1 no better off in that subgame for every situation. It turns out that this is an unnecessarily strong constraint. For example, 2♠7♥ is considered the worst hand in HUNL and should be folded immediately, which ends the game. Choosing any other action would result in an even bigger loss in expectation. Nevertheless, past subgame solving techniques would be concerned about P1 having 2♠7♥ in a subgame, which is unrealistic. Even if subgame solving resulted in a strategy that increased the value of 2♠7♥ a small amount in one subgame, that increase would not outweigh the cost of reaching the subgame (that is, the cost of not folding with 2♠7♥). Thus, P2 can allow the value of some "unimportant" P1 hands to increase in subgames, so long as the increase is small enough that it is still a mistake for P1 to reach the subgame with that hand. We accomplish this by increasing the alternative reward of P1 hands in the augmented subgame by the extra cost to P1 of reaching the subgame, that is, the size of the mistake P1 would have to make to reach that subgame with that hand. By increasing the alternative reward in the augmented subgame of these "unimportant" hands, P2 develops a strategy in the subgame that better defends against hands P1 might actually have (*4*).

(iii) Libratus crafts a unique strategy in response to opponent bets, rather than rounding it to the nearest size in the abstraction. The optimal response to a bet of $101 is different from the optimal response to a bet of $100, but the difference is likely minor. For that reason, rounding an opponent bet of $101 to $100 is reasonable. But the optimal response to a bet of $150 is likely significantly different from the response to a bet of $100 or a bet of $200. In principle one could simply increase the number of actions in the abstraction, perhaps by considering bets in increments of $10 rather than $100, so that the error from rounding is smaller. However, the size of the abstraction, and the time needed to solve it, increases prohibitively as more actions are added.

Therefore, rather than round to the nearest action, Libratus calculates a unique response in real time to off-tree actions, that is, an action taken by an opponent that is not in the abstraction. Libratus attempts to make the opponent no better off, no matter what hand the opponent might have, for having chosen the off-tree action rather than an action in the abstraction. It does this by generating and solving an augmented subgame following the off-tree action where the alternative payoff is the best in-abstraction action the opponent could have taken (the best action may differ across hands).

Libratus repeats this for every subsequent off-tree action in a process we call nested subgame solving (see Fig. 2). Later we provide experiments that demonstrate that this technique improves the worst-case performance of poker AIs by more than an order of magnitude compared to the best technique for rounding opponent actions to a nearby in-abstraction action.

(iv) Because the subgame is solved in real time, the abstraction in the subgame can also be decided in real time and change between hands. Libratus leverages this feature by changing, at the first point of subgame solving, the bet sizes it will use in that subgame and every subsequent subgame of that poker hand, thereby forcing the opponent to continually adapt to new bet sizes and strategies (*49*).

The authors of the poker AI DeepStack independently and concurrently developed an algorithm similar to nested subgame solving, which they call continual re-solving (*50*). In an Internet experiment, DeepStack defeated human professionals who are not specialists in HUNL. However, DeepStack was never shown to outperform prior publicly-available top AIs in head-to-head performance, whereas Libratus beats the prior leading HUNL poker AI Baby Tartanian8 by a wide margin, as we discuss later.

Like Libratus, DeepStack computes in real time a response to the opponent's specific bet and uses estimates rather than upper bounds on the opponent's values. It does not share Libratus's improvement of de-emphasizing hands the opponent would only be holding if she had made an earlier mistake, and does not share the feature of changing the subgame action abstraction between hands.

DeepStack solves a depth-limited subgame on the first two betting rounds by estimating values at the depth limit via a neural network. This allows it to always calculate real-time responses to opponent off-tree actions, while Libratus typically plays according to its pre-computed blueprint strategy in the first two rounds.

Because Libratus typically plays according to a pre-computed blueprint strategy on the first two betting rounds, it rounds an off-tree opponent bet size to a nearby in-abstraction action. The blueprint action abstraction on those rounds is dense in order to mitigate this weakness. In addition, Libratus has a unique self-improvement module to augment the blueprint strategy over time, which we now introduce.

Self-improvement

The third module of Libratus is the self-improver. It enhances the blueprint strategy in the background. It fills in missing branches in the blueprint abstraction and computes a game-theoretic strategy for those branches. In principle, one could conduct all such computations in advance, but the game tree is way too large for that to be feasible. To tame this complexity, Libratus uses the opponents' actual moves to suggest where in the game tree such filling is worthwhile.

The way machine learning has typically been used in game playing is to try to build an opponent model, find mistakes in the opponent's strategy (e.g., folding too often, calling too often, etc.), and exploit those mistakes (*51–53*). The downside is that trying to exploit the opponent opens oneself to being exploited. (A certain conservative family of exploitation techniques constitutes the sole exception to this downside (*51–53*).) For that reason, to a first approximation, Libratus did not do opponent exploitation. Instead, it used the data of the bet sizes that the opponents used to suggest which branches should be added to the blueprint, and it then computed game-theoretic strategies for those branches in the background.

In most situations that can occur in the first two betting rounds, real-time subgame solving as used in Libratus would likely not produce a better strategy than the blueprint, because the blueprint already uses no card abstraction in those rounds and conducting subgame solving in real time so early in the game tree would require heavy abstraction in the subgame. For these reasons, Libratus plays according to the precomputed blueprint strategy in these situations. In those rounds there are many bet sizes in the abstraction, so the error from rounding to a nearby size is small. Still, there is some error, and this could be reduced by including more bet sizes in the abstraction. In the experiment against human players described in the next section, Libratus analyzed the bet sizes in the first betting round most heavily used by its opponents in aggregate during each day of the competition. Based on the frequency of the opponent bet sizes and their distance from the closest bet size in the abstraction, Libratus chose k bet sizes for which it would try to calculate a response overnight (*54*). Each of those bet sizes for which reasonable convergence had been reached by the morning was then added to the blueprint strategy together with the newly-computed strategy following that bet size. In this way Libratus was able to progressively narrow its gaps as the competition proceeded by leveraging the humans' ability to find potential weaknesses. Furthermore, these fixes to its strategy are universal: they work against all opponents, not just the opponents that Libratus has faced.

Libratus's self-improvement comes in two forms. For one of them, when adding one of the k bet sizes, a default sibling bet size is also used during the equilibrium finding so as to not assume that the opponent necessarily only uses the bet size that will be added. For the other, a default bet size is not used. This can be viewed as more risky and even exploitative, but Libratus mitigates the risk by using that part of the strategy during play only if the opponent indeed uses that bet size most of the time (*4*).

Experimental evaluation

To evaluate the strength of the techniques used in Libratus, we first tested the overall approach of the AI on scaled-down variants of poker before proceeding to tests on full HUNL. These moderate-sized variants consisted of only two or three rounds of betting rather than four, and at most three bet sizes at each decision point. The smaller size of the games allowed us to precisely calculate exploitability, the distance from an optimal strategy. Performance was measured in milli-big blinds per hand (mbb/hand), the average number of big blinds won per 1,000 hands.

In the first experiment, we compared using no subgame solving, unsafe subgame solving (*42*) (in which a subgame is solved in isolation with no theoretical guarantees on performance), and safe subgame solving just once upon reaching the final betting round of the game. Both players were constrained to choosing among only two different bet sizes, so off-tree actions were not an issue in this first experiment. The results are shown in Table 1. In all cases, safe subgame solving reduced exploitability by more than a factor of 4 relative to no subgame solving. In one case, unsafe subgame solving led to even lower exploitability, while in another it increased exploitability by nearly an order of magnitude more than if no subgame solving had been used. This demonstrates that although unsafe subgame solving may produce strong strategies in some games, it may also lead to far worse performance. Safe subgame solving, in contrast, reduced exploitability in all games.

In the second experiment, we constructed an abstraction of a game which only includes two of the three available bet sizes. If the opponent played the missing bet size, the AI either used action translation [in which the bet is rounded to a nearby size in the abstraction; we compared against the leading action translation technique (*27*)], or nested subgame solving. The results are shown in Table 2. Nested subgame solving reduced exploitability by more than an order of magnitude relative to action translation.

Next we present experiments in full HUNL. After constructing Libratus, we tested the AI against the prior leading HUNL poker AI, our 2016 bot Baby Tartanian8, which had defeated all other poker AIs with statistical significance in the most recent ACPC (*55*). We report average win rates followed by the 95% confidence interval. Using only the raw blueprint strategy, Libratus lost to Baby Tartanian8 by 8 ± 15 mbb/hand. Adding state-of-the-art post-processing on the 3rd and 4th betting rounds (*31*), such as eliminating low-probability actions that are likely only positive owing to insufficient time to reach convergence, led to the Libratus blueprint strategy defeating Baby Tartanian8 by 18 ± 21 mbb/hand. Eliminating low-probability actions empirically leads to better performance against non-adjusting AIs. However, it also increases the exploitability of the AI because its strategy becomes more predictable. The full Libratus agent did not use post-processing on the third and fourth betting rounds. On the first two rounds, Libratus primarily used a new, more robust, form of post-processing (*4*).

The next experiment evaluated nested subgame solving (with no post-processing) using only actions that are in Baby Tartanian8's action abstraction. Libratus won by 59 ± 28 mbb/hand (*56*). Finally, applying the nested subgame solving structure used in the competition resulted in Libratus defeating Baby Tartanian8 by 63 ± 28 mbb/hand. The results are shown in Table 3. In comparison, Baby Tartanian8 defeated the next two strongest AIs in the ACPC by 12 ± 10 mbb/hand and 24 ± 20 mbb/hand.

Finally, we tested Libratus against top humans. In January 2017, Libratus played against a team of four top HUNL specialist professionals in a 120,000-hand Brains vs. AI challenge match over 20 days. The participants were Jason Les, Dong Kim, Daniel McCauley, and Jimmy Chou. A prize pool of $200,000 was allocated to the four humans in aggregate. Each human was guaranteed $20,000 of that pool. The remaining $120,000 was divided among them based on how much better the human did against Libratus than the worst-performing of the four humans. Libratus decisively defeated the humans by a margin of 147 mbb/hand, with 99.98% statistical significance and a p-value of 0.0002 (if the hands are treated as independent and identically distributed), see Fig. 3 (*57*). It also beat each of the humans individually.

Conclusions

Libratus presents an approach that effectively addresses the challenge of game-theoretic reasoning under hidden information in a large state space. The techniques that we developed are largely domain independent and can thus be applied to other strategic imperfect-information interactions, including non-recreational applications. Owing to the ubiquity of hidden information in real-world strategic interactions, we believe the paradigm introduced in Libratus will be important for the future growth and widespread application of AI.

REFERENCES AND NOTES

1. J. Schaeffer, *One Jump Ahead: Challenging Human Supremacy in Checkers* (Springer, 1997).
2. M. Campbell, A. J. Hoane Jr., F.-H. Hsu, Deep Blue. *Artif. Intell.* **134**, 57–83 (2002). doi:10.1016/S0004-3702(01)00129-1
3. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016). doi:10.1038/nature16961 Medline
4. See supplementary materials for more details.
5. J. Nash, "Non-cooperative games," thesis, Princeton University (1950).
6. J. F. Nash, L. S. Shapley, *Contributions to the Theory of Games*, H. W. Kuhn, A. W. Tucker, Eds. (Princeton Univ. Press, 1950), vol. 1, pp. 105–116.
7. D. A. Waterman, Generalization learning techniques for automating the learning of heuristics. *Artif. Intell.* **1**, 121–170 (1970). doi:10.1016/0004-3702(70)90004-4
8. J. Shi, M. Littman, in *CG '00: Revised Papers from the Second International Conference on Computers and Games* (Springer, 2002), pp. 333–345.
9. D. Billings *et al*., in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)* (Morgan Kaufmann Publishers, San Francisco, 2003), pp. 661–668.
10. A. Gilpin, T. Sandholm, in *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2005), pp. 1684–1685.
11. M. Bowling, N. Burch, M. Johanson, O. Tammelin, Heads-up limit hold'em poker is solved. *Science* **347**, 145–149 (2015). doi:10.1126/science.1259433 Medline
12. Libratus is Latin and means balanced (for approximating Nash equilibrium) and forceful (for its powerful play style and strength).
13. An imperfect-information subgame (which we refer to simply as a subgame) is defined differently than how a subgame is usually defined in game theory. The usual definition requires that a subgame starts with the players knowing the exact state of the game, that is, no information is hidden from any player. Here, an imperfect-information subgame is determined by information that is common knowledge to the players. For example, in poker, a subgame is defined by the sequence of visible board cards and actions the players have taken so far. Every possible combination of private cards—that is, every node in the game tree which is consistent with the common knowledge—is a root of this subgame. Any node that descends from a root node is also included in the subgame. A formal definition is provided in the supplementary material.
14. N. Burch, M. Johanson, M. Bowling, in *AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2014), pp. 602–608.
15. M. Moravcik, M. Schmid, K. Ha, M. Hladik, S. Gaukrodger, in *AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2016).
16. E. Jackson, in *AAAI Workshop on Computer Poker and Imperfect Information* (AAAI Press, 2014).
17. M. Zinkevich, M. Johanson, M. H. Bowling, C. Piccione, in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)* (Neural Information Processing Systems Foundation, Inc., 2007), pp. 1729–1736.
18. Y. Nesterov, Excessive gap technique in nonsmooth convex minimization. *SIAM J. Optim.* **16**, 235–249 (2005). doi:10.1137/S1052623403422285
19. S. Hoda, A. Gilpin, J. Peña, T. Sandholm, Smoothing techniques for computing Nash equilibria of sequential games. *Math. Oper. Res.* **35**, 494–512 (2010). doi:10.1287/moor.1100.0452
20. A. Gilpin, J. Peña, T. Sandholm, First-order algorithm with $O(\ln(1/\epsilon))$ convergence for ϵ-equilibrium in two-person zero-sum games. *Math. Program.* **133**, 279–298 (2012). doi:10.1007/s10107-010-0430-2.
21. C. Kroer, K. Waugh, F. Klnç-Karzan, T. Sandholm, in *Proceedings of the ACM Conference on Economics and Computation (EC)* (ACM, New York, 2017).
22. O. Tammelin, N. Burch, M. Johanson, M. Bowling, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (AAAI Press, 2015), pp. 645–652.
23. The version of HUNL that we refer to, which is used in the Annual Computer Poker Competition, allows bets in increments of $1, with each player having $20,000 at the beginning of a hand.
24. M. Johanson, "Measuring the size of large no-limit poker games," (Technical Report, Univ. of Alberta Libraries, 2013).
25. A. Gilpin, T. Sandholm, T. B. Sørensen, in *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2008), vol. 2, pp. 911–918.
26. D. Schnizlein, M. Bowling, D. Szafron, in *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence* (AAAI Press, 2009), pp. 278–284.
27. S. Ganzfried, T. Sandholm, in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* (AAAI Press, 2013), pp. 120–128.
28. Annual Computer Poker Competition; www.computerpokercompetition.org.
29. N. Brown, T. Sandholm, in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2014), pp. 594–601.
30. N. Brown, T. Sandholm, in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)* (AAAI Press, 2016), pp. 4238–4239.
31. N. Brown, S. Ganzfried, T. Sandholm, in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2015), pp. 7–15.
32. N. Brown, S. Ganzfried, T. Sandholm, in *AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2015), pp. 4270–4271.
33. M. Johanson, N. Burch, R. Valenzano, M. Bowling, in *Proceedings of the 2013 International Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2013), pp. 271–278.
34. M. Lanctot, K. Waugh, M. Zinkevich, M. Bowling, in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)* (Neural Information Processing Systems Foundation, Inc., 2009), pp. 1078–1086.
35. R. Gibson, M. Lanctot, N. Burch, D. Szafron, M. Bowling, in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (AAAI Press, 2012), pp. 1355–1361.
36. M. Johanson, N. Bard, M. Lanctot, R. Gibson, M. Bowling, in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2012), vol. 2, pp. 837–846.

37. R. Gibson, "Regret minimization in games and the development of champion multiplayer computer poker-playing agents," thesis, University of Alberta (2014).
38. There are a number of theoretically correct ways to choose actions on the basis of their regrets. The most common is regret matching, in which an action is chosen in proportion to its positive regret (*58*). Another common choice is hedge (*59*, *60*).
39. An action a with regret $R(a)$ that is below a threshold C (where C is negative) is sampled with probability $K/[K + C - R(a)]$, where K is a positive constant. There is additionally a floor on the sample probability. This sampling is only done for about the last half of iterations to be run; the first half is conducted using traditional external-sampling MCCFR. Other formulas can also be used.
40. K. Waugh *et al*., in *Symposium on Abstraction, Reformulation, and Approximation (SARA)* (AAAI Press, 2009).
41. M. Johanson, N. Bard, N. Burch, M. Bowling, in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (AAAI Press, 2012), pp. 1371–1379.
42. S. Ganzfried, T. Sandholm, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (International Foundation for Autonomous Agents and Multiagent Systems, 2015), pp. 37–45.
43. N. Brown, T. Sandholm, *Adv. Neural Inf. Process. Syst.* **30**, 689–699 (2017).
44. In Libratus, we considered "sufficiently small" to be situations where no additional bets or raises could be made.
45. Despite lacking theoretical guarantees, unsafe subgame solving empirically performs well in certain situations and requires less information to be precomputed. For this reason, Libratus uses it once upon first reaching the third betting round, while using safe subgame solving in all subsequent situations.
46. We solved augmented subgames using a heavily optimized form of the CFR+ algorithm (*22*, *61*) because of the better performance of CFR+ in small games where a precise solution is desired. The optimizations we use keep track of all possible P1 hands rather than dealing out a single one at random.
47. Note that the theorem only assumes perfect recall in the actual game, not in the abstraction that is used for computing a blueprint strategy. Furthermore, applying Estimated-Maxmargin assumes that that subroutine maximizes the minimum margin; a sufficient condition for doing so is that there is no abstraction in the subgame.
48. Indeed, the original purpose of safe subgame solving was merely to reduce space usage by reconstructing subgame strategies rather than storing them.
49. Specifically, Libratus increased or decreased all its bet sizes by a percentage chosen uniformly at random between 0 and 8%.
50. M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, M. Bowling, DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* **356**, 508–513 (2017). doi:10.1126/science.aam6960 Medline
51. D. Billings, D. Papp, J. Schaeffer, D. Szafron, in *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (AAAI Press, 1998), pp. 493–499.
52. S. Ganzfried, T. Sandholm, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (International Foundation for Autonomous Agents and Multiagent Systems, 2011).
53. S. Ganzfried, T. Sandholm, Safe opponent exploitation. *ACM Transaction on Economics and Computation (TEAC)* **3**, 1–28 (2015). doi:10.1145/2716322
54. Based on the available computing resources, we chose $k = 3$ so that the algorithm could typically fix three holes to reasonable accuracy in 24 hours.
55. Baby Tartanian8 and all other AIs in the ACPC are available to ACPC participants for benchmarking.
56. Baby Tartanian8 uses action translation in response to bet sizes that are not in its action abstraction. Our experiments above demonstrated that action translation performs poorly compared to subgame solving. Using only bet sizes in Baby Tartanian8's abstraction disentangles the effects of action translation from the improvement of nested subgame solving. Baby Tartanian8 still used actions that were not in Libratus's abstraction, and therefore the experiments can be considered conservative.
57. Because both the humans and the AI adapted over the course of the competition, treating the hands as independent is not entirely inappropriate. We include confidence figures to provide some intuition for the variance in HUNL. In any case, 147 mbb/hand over 120,000 hands is considered a massive and unambiguous victory in HUNL.
58. S. Hart, A. Mas-Colell, A simple adaptive procedure leading to correlated equilibrium. *Econometrica* **68**, 1127–1150 (2000). doi:10.1111/1468-0262.00153
59. N. Littlestone, M. K. Warmuth, The weighted majority algorithm. *Inf. Comput.* **108**, 212–261 (1994). doi:10.1006/inco.1994.1009
60. Y. Freund, R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* **55**, 119–139 (1997). doi:10.1006/jcss.1997.1504
61. M. Johanson, K. Waugh, M. Bowling, M. Zinkevich, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (AAAI Press, 2011), pp. 258–265.
62. L. Kocsis, C. Szepesvári, in *European Conference on Maching Learning (ECML)* (Springer, 2006), pp. 282–293.
63. R. Coulom, *Computers and Games* (Springer, 2007), pp. 72–83.
64. D. E. Knuth, R. W. Moore, An analysis of alpha-beta pruning. *Artif. Intell.* **6**, 293–326 (1975). doi:10.1016/0004-3702(75)90019-3
65. J. F. Nash, Equilibrium points in *n*-person games. *Proc. Natl. Acad. Sci. U.S.A.* **36**, 48–49 (1950). doi:10.1073/pnas.36.1.48 Medline
66. N. Brown, T. Sandholm, in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)* (2015), pp. 1972–1980.
67. N. Brown, T. Sandholm, in *International Conference on Machine Learning* (Proceedings of Machine Learning Research, 2017).
68. S. Ganzfried, T. Sandholm, K. Waugh, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (International Foundation for Autonomous Agents and Multiagent Systems, 2012), pp. 871–878.

ACKNOWLEDGMENTS

This material is based on research supported by the National Science Foundation under grants IIS-1718457, IIS-1617590, and CCF-1733556, and by the ARO under award W911NF-17-1-0082, as well as XSEDE computing resources provided by the Pittsburgh Supercomputing Center. The Brains vs. AI competition was sponsored by Carnegie Mellon University, Rivers Casino, GreatPoint Ventures, Avenue4Analytics, TNG Technology Consulting, Artificial Intelligence, Intel, and Optimized Markets, Inc. We thank Ben Clayman for computing statistics of the play of our AIs against humans. The data presented in this paper are shown in the main text and supplementary material. Additional data can be obtained from the corresponding author upon request. Because HUNL poker is played commercially, the risk associated with releasing the code outweighs the benefits. To aid reproducibility, we have included the pseudo-code for the major components of our program in (*4*). The technology has been exclusively licensed to Strategic Machine, Inc., and the authors have ownership interest in the company.

SUPPLEMENTARY MATERIALS

www.sciencemag.org/cgi/content/full/science.aao1733/DC1
Supplementary text
Figs. S1 and S2
Table S1
References (*62*–*68*)

22 June 2017; accepted 12 December 2017
Published online 17 December 2017
10.1126/science.aao1733

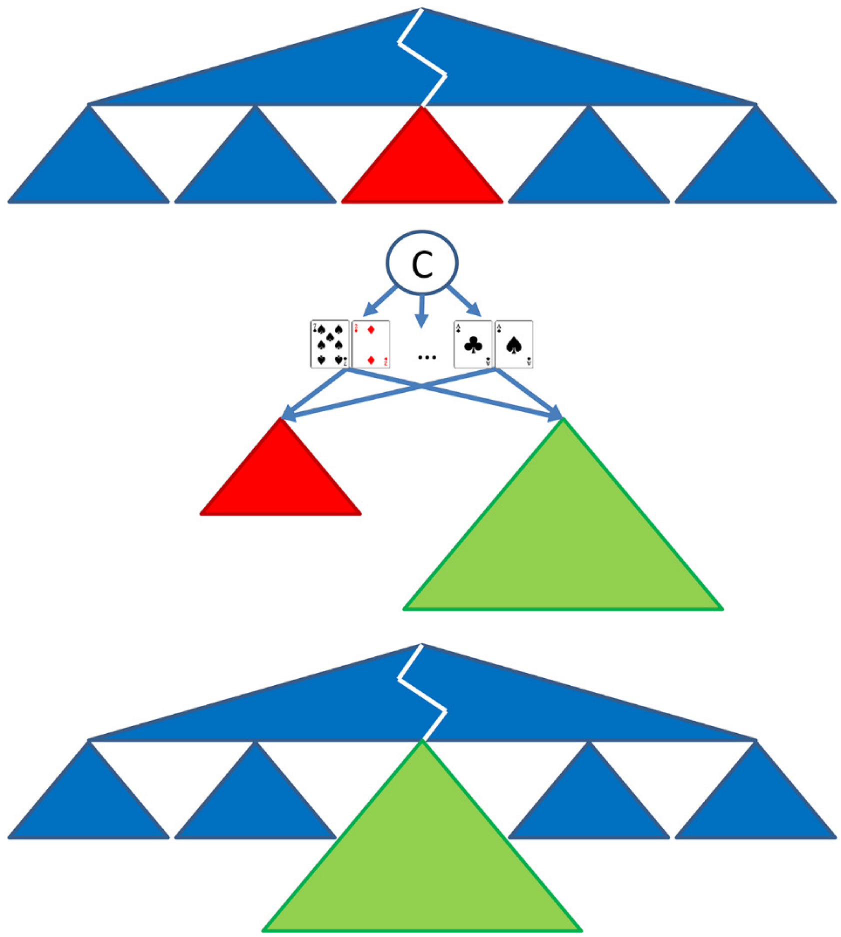

Fig. 1. Subgame solving. Top: A subgame is reached during play. Middle: A more detailed strategy for that subgame is determined by solving an augmented subgame, in which on each iteration the opponent is dealt a random poker hand and given the choice of taking the expected value of the old abstraction (red), or of playing in the new, finer-grained abstraction (green) where the strategy for both players can change. This forces Libratus to make the finer-grained strategy at least as good as in the original abstraction against every opponent poker hand. Bottom: The new strategy is substituted in place of the old one.

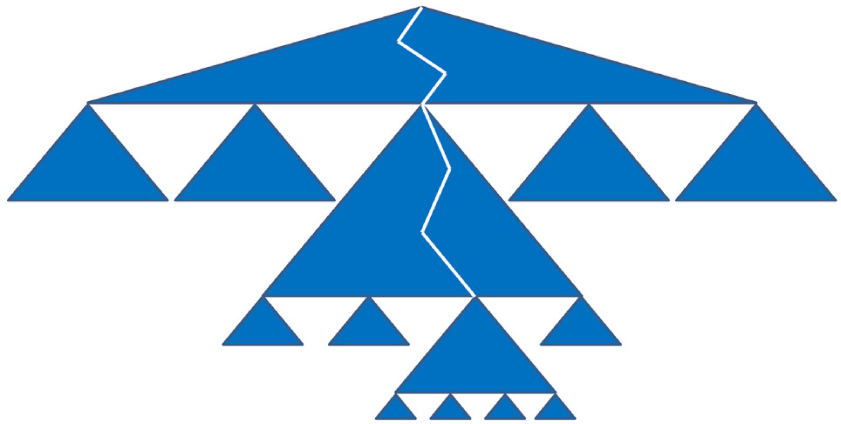

Fig. 2. A visualization of nested subgame solving. Every time a subgame is reached during play, a more detailed abstraction is constructed and solved just for that subgame, while fitting its solution within the overarching blueprint strategy.

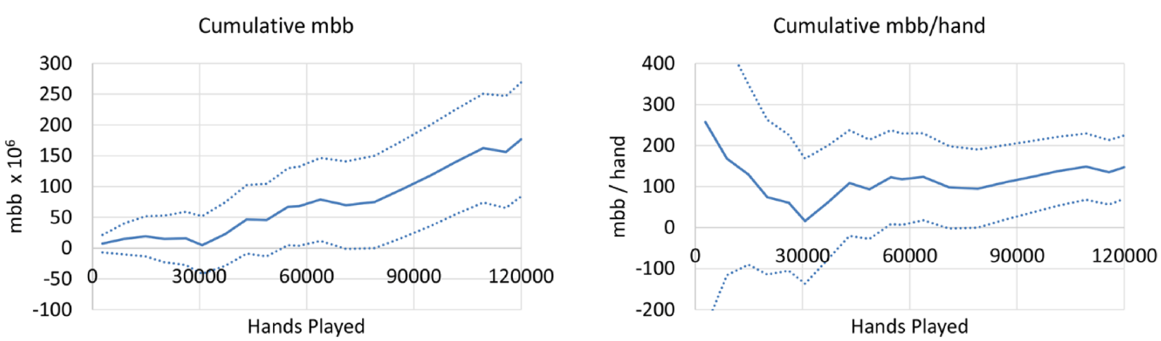

Fig. 3. Libratus performance against top humans. Shown are the results of the 2017 Brains vs. AI competition. The 95% confidence intervals (if the hands are treated as independent and identically distributed) are shown as dotted lines.

Table 1. Exploitability of subgame solving techniques on smaller poker variants.

Simplified Game:	Small 2-Round	Large 2-Round Hold'em	3-Round Hold'em
No subgame solving	91.3 mbb/hand	41.3 mbb/hand	346 mbb/hand
Unsafe subgame solving	5.51 mbb/hand	397 mbb/hand	79.3 mbb/hand
Safe subgame solving	22.6 mbb/hand	9.84 mbb/hand	72.6 mbb/hand

Table 2. Exploitability of nested subgame solving. Shown is the comparison to no nested subgame solving (which instead uses the leading action translation technique) in a small poker variant.

	Exploitability
No nested subgame solving	1,465 mbb/hand
Nested unsafe subgame solving	148 mbb/hand
Nested safe subgame solving	119 mbb/hand

Table 3. Head-to-head performance of Libratus. Shown are results for the Libratus blueprint strategy as well as forms of nested subgame solving against Baby Tartanian8 in HUNL.

	Performance against *Baby Tartanian8*
Blueprint	−8 ±15 mbb/hand
Blueprint with post-processing	18 ± 21 mbb/hand
On-tree nested subgame solving	59 ± 28 mbb/hand
Full nested subgame solving	63 ± 28 mbb/hand