Fast Payoff Matrix Sparsification Techniques for Structured Extensive-Form Games

Gabriele Farina[1], **Tuomas Sandholm**[1,2,3,4]
[1] Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213
[2] Strategy Robot, Inc.
[3] Optimized Markets, Inc.
[4] Strategic Machine, Inc.
{gfarina,sandholm}@cs.cmu.edu

Abstract

The practical scalability of many optimization algorithms for large extensive-form games is often limited by the games' huge payoff matrices. To ameliorate the issue, Zhang and Sandholm recently proposed a sparsification technique that factorizes the payoff matrix $\boldsymbol{A}$ into a sparser object $\boldsymbol{A} = \hat{\boldsymbol{A}} + \boldsymbol{U}\boldsymbol{V}^\top$, where the total combined number of nonzeros of $\hat{\boldsymbol{A}}$, $\boldsymbol{U}$, and $\boldsymbol{V}$ is significantly smaller. Such a factorization can be used in place of the original payoff matrix in many optimization algorithm, such as interior-point and second-order methods, thus increasing the size of games that can be handled. Their technique significantly sparsifies poker (end)games, standard benchmarks used in computational game theory, AI, and more broadly. We show that the existence of extremely sparse factorizations in poker games can be tied to their particular *Kronecker-product* structure. We clarify how such structure arises and introduce the connection between that structure and sparsification. By leveraging such structure, we give two ways of computing strong sparsifications of poker games (as well as any other game with a similar structure) that are i) orders of magnitude faster to compute, ii) more numerically stable, and iii) produce a dramatically smaller number of nonzeros than the prior technique. Our techniques enable—for the first time—effective computation of high-precision Nash equilibria and strategies subject to constraints on the amount of allowed randomization. Furthermore, they significantly speed up parallel first-order game-solving algorithms; we show state-of-the-art speed on a GPU.

1 Introduction

Certain important quantities of interest in computational game theory can be expressed as the solution to a linear program (LP) and therefore—in principle—solved for by any algorithm for linear optimization. The practice is more nuanced. The size of the LP is usually dominated by the payoff matrix of the game, that is, the matrix of payoffs for each of the possible terminal states of the game. Correspondingly, in large extensive-form games, most out-of-the-box algorithms for linear programming—such as interior point methods and the simplex method—are unviable. This has historically led the research community in computational game theory to develop specialized (as opposed to applicable to any linear program) algorithms—usually first-order methods—that avoid the need for representing the payoff matrix explicitly. Among these, the most successful examples include the CFR algorithm (Zinkevich et al. 2007) and its modern variants (Tammelin 2014; Moravčík et al. 2017; Brown and Sandholm 2017b,a, 2019a,b; Davis, Waugh, and Bowling 2019; Farina, Kroer, and Sandholm 2021b; Morrill et al. 2021), and methods based on accelerated first-order methods such as EGT (Nesterov 2005; Hoda et al. 2010; Kroer, Farina, and Sandholm 2018; Farina, Kroer, and Sandholm 2021a) and Mirror Prox (Nemirovski 2004; Kroer 2019; Farina, Kroer, and Sandholm 2021a), which are able to scale to large two-player extensive-form games and compute approximate Nash equilibria for moderate approximation gaps. However, there exist certain applications where currently only LP and linear integer programming technology provide suitable guarantees. For example, the only scalable method for computing sequentially-rational equilibria depends on the ability to find high-precision Nash equilibria, a task that can currently only be achieved using LP technology. Another example is computation of strategies subject to constraints such as support size, sparsity, or amount of randomization, an optimization problem that can easily be expressed via integer linear programming.

In a recent paper, Zhang and Sandholm (2020) propose a technique to factorize the payoff matrix of any two-player extensive-form game into a low-rank decomposition (called a *sparsification*) such that the number of nonzeros required in the decomposition is significantly smaller than the number of nonzeros in the original payoff matrix. They show that such a factorization can then be used in place of the original payoff matrix in certain LPs, thereby increasing the game size that LP technology is able to handle. While their sparsification technique is able to typically reduce the number of nonzeros by a factor of 2–3, a notable empirical finding in their evaluation is the dramatic reduction in the number of nonzeros—close to two orders of magnitude—that their heuristic achieves in two-player poker endgames. That is important due to the central role of poker in imperfect-information game solving. Poker variants have been the standard canonical benchmarks in game theory since the introduction of the most seminal solution concept, Nash equilibrium, in 1950 (Nash 1950a; Kuhn 1950). Poker cap-

tures the essence of private information and strategic, game-theoretic deception and reasoning. In fact, in Nash's dissertation, the only application was poker (Nash 1950b). In the ensuing decades, larger and larger poker variants were tackled in AI (Waterman 1970) and operations research (Zadeh 1977). Then around year 2000, poker was recognized as the main challenge problem for imperfect-information game solving in AI (Billings et al. 2002). Hundreds of papers have been published on it, the AAAI Annual Computer Poker Competition was organized, and superhuman AI performance has been achieved (Bowling et al. 2015; Brown and Sandholm 2017b, 2019b). This has dramatically pushed the boundary of imperfect-information game-solving technology. As many questions in the field remain open (for example, the computation of interpretable, sparse, collusive, or sequentially-rational strategies), we are convinced that poker will continue to play a fundamental role as the gold standard in imperfect-information games for decades to come.

We show that the existence of extremely sparse factorizations in poker games can be tied to their particular *Kronecker-product* structure. The existence of that structure was mentioned by Hoda et al. (2010) solely for the purpose of reducing memory footprint of their first-order method for Nash equilibrium, and Johanson et al. (2011) use essentially the same structure (without the Kroenecker representation) to speed up best-response computation in poker. We clarify how Kroenecker-product structure arises and, most importantly, introduce the connection between that structure and sparsification. By leveraging the Kronecker-product structure directly, we give two ways of computing strong sparsifications of poker games (as well as any other game with a similar structure). We show that our sparsification techniques are i) orders of magnitude faster to compute, ii) more numerically stable, and iii) produce a dramatically smaller number of nonzeros than the general algorithm by Zhang and Sandholm (2020). Our sparsification techniques enable—for the first time—effective computation of high-precision Nash equilibria and strategies subject to constraints on the amount of allowed randomization. Furthermore, they significantly speed up parallel first-order game-solving algorithms; we show state-of-the-art speed on GPU.

Weaknesses While our techniques apply to all games with a Kronecker-product structure (that is, whose payoff matrix can be expressed as a sum of Kronecker products), currently the only games with practical relevance that are known to exhibit a Kronecker-product structure are poker games. That said, as many questions in the field remain open (for example, the computation of interpretable, sparse, collusive, or sequentially-rational strategies), we believe that poker will continue to play a fundamental role as the gold standard in imperfect-information games for decades to come.

As we show, our techniques have the concrete potential to help make a dent on those important questions by enabling one to scale up existing optimization methods—essentially for free—by replacing the payoff matrix of the game with its sparsified counterpart.

2 Payoff Matrix Sparsification and Its Applications

Extensive-form games are played on a game tree and can capture both sequential and simultaneous moves, stochastic events (such as a roll of the dice, or drawing a random card from a shuffled deck) as well as private information. A strategy for a generic Player i in an extensive-form game is an assignment of probability to each of the player's *sequences*—that is, sequence of actions that the player can take starting from the root of the game tree. Just like in normal-form games, the outcomes of a two-player extensive-form game can be arranged compactly into a *payoff matrix* $\boldsymbol{A}$, whose rows and columns are indexed over all sequences of the two players. Specifically, let z be an outcome (terminal state) of the game tree, let u be the payoff assigned to Player i by that outcome, and let σ_i, σ_j be the sequences for Player i and her opponent, respectively, corresponding to z. Finally, let c be the product of the probability of all stochastic events on the path from the root of the game tree to z. Then, Player i's payoff matrix contains, on the row corresponding to sequence σ_i and column corresponding to σ_j, a payoff equal to $u \cdot c$. For the purposes of this paper, a *sparsification* of the payoff matrix $\boldsymbol{A}$ of a game will be defined as an expression of the form

$$\boldsymbol{A} = \hat{\boldsymbol{A}} + \boldsymbol{U}\boldsymbol{M}^{-1}\boldsymbol{V}^{\top}, \tag{1}$$

for suitable matrices $\hat{\boldsymbol{A}}, \boldsymbol{U}, \boldsymbol{M}$ and $\boldsymbol{V}$, where $\boldsymbol{M}$ is an invertible triangular matrix. The expression in (1) is more general than the one considered by Zhang and Sandholm (2020), which corresponds to the case where $\boldsymbol{M}$ is the identity matrix. We will show in Section 5 how the flexibility afforded by the matrix $\boldsymbol{M}$ translates into better performance. Given a sparsification of $\boldsymbol{A}$, we will refer to its *size* as the sum of the number of nonzeros of the matrices $\hat{\boldsymbol{A}}, \boldsymbol{U}, \boldsymbol{V}$ and $\boldsymbol{M}$. A "good" sparsification is one whose size is significantly smaller than the number of nonzeros of the original matrix $\boldsymbol{A}$. We will investigate three main applications of payoff matrix sparsification.

1. Linear Programming and High-Precision Nash Equilibrium Strategies. It is well-known that a Nash equilibrium strategy for a player in a two-player zero-sum perfect-recall extensive-form game can be expressed as the solution to an LP by using the *sequence-form representation* (von Stengel 1996; Koller, Megiddo, and von Stengel 1996; Romanovskii 1962). Specifically, given the payoff matrix $\boldsymbol{A}$ (say, for Player 1), a Nash equilibrium strategy for that player is the solution to the LP on the left of (2), where the matrices $\boldsymbol{F}_i$ and vectors $\boldsymbol{f}_i$ (for $i \in \{1, 2\}$) are very sparse and define the sequence-form constraints for Player 1 and 2, respectively. With a sparsification of $\boldsymbol{A}$, the LP on the left of (2) can be rewritten as the one on the right, trading the number of nonzeros of $\boldsymbol{A}$ for the size of the sparsification on the right. When the size of the sparsification is much smaller than the number of nonzeros of $\boldsymbol{A}$, the LP on the right is significantly sparser, and can therefore be solved much faster (or at all, in large games) by LP technology. That enables the computation of Nash equilibrium strategies at a high level of precision, a task that is infeasible for it-

$$\begin{cases} \max\ \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{① } \boldsymbol{A}^\top \boldsymbol{x} - \boldsymbol{F}_2^\top \boldsymbol{v} \geq \boldsymbol{0} \\ \text{② } \boldsymbol{F}_1 \boldsymbol{x} = \boldsymbol{f}_1 \\ \text{③ } \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{v} \text{ free} \end{cases} \rightarrow \begin{cases} \max\ \boldsymbol{f}_2^\top \boldsymbol{v} \\ \text{① } \hat{\boldsymbol{A}}^\top \boldsymbol{x} - \boldsymbol{F}_2^\top \boldsymbol{v} + \boldsymbol{V}\boldsymbol{w} \geq \boldsymbol{0} \\ \text{② } \boldsymbol{F}_1 \boldsymbol{x} = \boldsymbol{f}_1 \\ \text{③ } \boldsymbol{U}^\top \boldsymbol{x} - \boldsymbol{M}^\top \boldsymbol{w} = \boldsymbol{0} \\ \text{④ } \boldsymbol{x} \geq \boldsymbol{0}, \boldsymbol{v} \text{ free}, \boldsymbol{w} \text{ free.} \end{cases} \tag{2}$$

erative first-order methods (such as CFR (Zinkevich et al. 2007) and EGT (Hoda et al. 2010; Kroer, Farina, and Sandholm 2018)). One immediate application of computing Nash equilibrium strategies at that level of precision is the ability to compute the *exact* value of the game. Another important reason is that the computation of optimal, basic strategies (that is, vertices of the LP) represents a fundamental building block in the computation of sequentially-rational equilibrium refinements (Farina, Gatti, and Sandholm 2018). In Section 2 we show that our sparsification techniques enable one to compute high-precision Nash equilibrium strategies in games significantly larger than what was possible with the sparsification technique of Zhang and Sandholm (2020).

2. Integer Programming and Least-Exploitabile Deterministic Strategies. Deterministic strategies can be deployed without the need for randomization—at which humans are notoriously bad—and are arguably more interpretable than randomized strategies. How much randomization is needed to play optimally in poker is a long-standing open question. (Some early work on simplified models has suggested that not much randomization is needed (Chen and Ankenman 2006; Ganzfried and Sandholm 2010).) Our sparsification techniques help scale the computation of strategies subject to constraints on the amount of required randomization. For instance, a least-exploitable deterministic strategy can be computed as the solution to the integer program obtained from either formulation in (2) by replacing the constraint $\boldsymbol{x} \geq \boldsymbol{0}$ with the constraints that $\boldsymbol{x}$ be a vector of binary variables. In large games, even state-of-the-art commercial integer programming technology cannot even remotely scale up to the size of the unsparsified formulation. Instead, in Section 5 we will show that the same formulation sparsified with our techniques enables—to our knowledge, for the first time—the computation of provably near-optimal deterministic strategies. We will also measure how much less value a deterministic player can guarantee herself—a metric we coin *price of determinism*. We find that in the real no-limit Texas hold'em endgames we test on, the price of determinism is minimal: deterministic strategies extract at least 98.26% of the value of the game in all cases. Given the benefits of deterministic strategies (such as lower memory requirement, no need to randomize, higher interpretabilty, and ease of deployment by humans), we believe this to be an interesting positive experimental outcome on a long-standing research question that also warrants further investigation.

3. First-Order Methods and Highly-Parallel Gradient Computation. First-order methods that compute approximate Nash equilibrium strategies—such as CFR (Zinkevich et al. 2007) and EGT (Hoda et al. 2010; Kroer, Farina, and Sandholm 2018)—require, as an intermediate step at each iteration, the evaluation of the gradient of the utility function, which can be computed via a sparse matrix-vector multiplication between the payoff matrix $\boldsymbol{A}$ and the strategy $\boldsymbol{x}$ of a player. Given a sparsification of $\boldsymbol{A}$, the following is a natural algorithm for computing $\boldsymbol{A}\boldsymbol{x}$: first, compute the product $\boldsymbol{y} := \boldsymbol{V}^\top \boldsymbol{x}$; then, solve the sparse triangular system $\boldsymbol{M}\boldsymbol{z} = \boldsymbol{y}$, solving for $\boldsymbol{z}$ (we skip this step when $\boldsymbol{M}$ is the identity matrix, and instead immediately let $\boldsymbol{z} = \boldsymbol{y}$); then, multiply the solution $\boldsymbol{z}$ of the triangular system by $\boldsymbol{U}$, computing $\boldsymbol{w} := \boldsymbol{U}\boldsymbol{z}$; finally, sum the sparse matrix-vector product $\hat{\boldsymbol{A}}\boldsymbol{x}$ to $\boldsymbol{w}$. Each of the matrix-vector products involved requires a number of operations proportional to the number of nonzeros of the matrix. Furthermore, since $\boldsymbol{M}$ is triangular, $\boldsymbol{z}$ can be computed in time proportional to the number of nonzeros in $\boldsymbol{M}$. So, the number of floating-point operations required by the algorithm is proportional to the size of the sparsification. Given the wide availability of highly-tuned libraries for sparse matrix-vector multiplication both for CPUs and GPUs, the method we have just described enables an extremely concise and efficient implementation of the gradient of the utility function of sparsified games, which can easily rival specialized combinatorial algorithms (Johanson et al. 2011).

3 Kronecker-Product Structure of Poker Games

In this section, we illustrate and formalize a particular combinatorial structure—which we refer to as *Kronecker-product structure*—that poker games possess. Only a basic working knowledge of poker is needed to follow this section. In the appendix we describe the basic rules of poker. The term *Kronecker-product structure* refers to the fact that the payoff matrix can be expressed as (a sum of) terms of the form

$$\boldsymbol{P} \otimes \boldsymbol{Q} := \begin{bmatrix} P_{11}\boldsymbol{Q} & \cdots & P_{1n}\boldsymbol{Q} \\ \vdots & \ddots & \vdots \\ P_{m1}\boldsymbol{Q} & \cdots & P_{mn}\boldsymbol{Q} \end{bmatrix} \in \mathbb{R}^{(mr)\times(ns)},$$

for appropriate matrices $\boldsymbol{P} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{Q} \in \mathbb{R}^{r\times s}$ and arbitrary dimensions m, n, r, s.

In this section, we shed light on how this structure arises, by focusing on the endgame that begins immediately after the last (aka. river) card is revealed—called a *river endgame*. The observations we will make about river endgames in this subsection apply more generally to the endgame that begins immediately after the turn and the flop, as well as the full poker game. We will conventionally refer to the first mover in the endgame (that is, the "small blind" player) as 'Player 1', and to the second mover (the "big blind" player) as 'Player 2'. We will focus on computing the payoff matrix for Player 1; the payoff matrix for Player 2 is completely analogous.

In a river endgame, all community cards have already been revealed, and the two players engage in a single round of betting before the endgame ends. To fully describe a particular instance of a river endgame, the following quanti-

ties must be given: (i) The collection B of five community cards (the *board*) that have been drawn; (ii) Initial stack sizes (s_1, s_2), the amount of money that Player 1 and 2, respectively, possess in their stack at the beginning of the endgame; (iii) Initial pot contribution c, the amount of money that have been contributed to the pot by Player 1 and 2, prior to the endgame; (iv) Two belief distributions $\mu_i : \mathcal{H}_i \to [0, 1]$, one for each player $i \in \{1, 2\}$, assigning a probability distribution to each possible hand of the players.[1] The river endgame is an extensive-form game of its own, where at the root of the game tree a chance node assigns private hands $(h_1, h_2) \in \mathcal{H}_1 \times \mathcal{H}_2$ compatible with B (that is, so that when putting together the hands and the board, no card appears more than once) to each player according to the distribution

$$\pi(h_1, h_2) = \frac{1}{\beta} \begin{cases} 0 & \text{if } h_1, h_2, B \text{ are incompatible} \\ \mu_1(h_1)\mu_2(h_2) & \text{otherwise,} \end{cases}$$

where β is the appropriate normalization constant so that $\sum_{h_1,h_2} \pi(h_1, h_2) = 1$. Then, the game proceeds with one betting round (with the standard mechanics recalled in Appendix A), which can either end with a player folding, or with a showdown. The actions that the players can take in the betting round is the same, *regardless of their private hands*. In other words, the subtrees rooted under each possible outcome of the root chance nodes (which corresponds to an assignment of hands for each player), are all equal. To study the combinatorial properties of the game tree corresponding to the river endgame, it is then natural to only focus on *one*, generic such subtree, which we call the *skeleton* of the river endgame. Figure 1 depicts the skeleton of a river endgame for a very coarse betting abstraction. The payoff matrix $\boldsymbol{A}$ for any player can be expressed as a block matrix $[\boldsymbol{A}_{h_1,h_2}]_{h_1 \in \mathcal{H}_1, h_2 \in \mathcal{H}_2}$, where each block $\boldsymbol{A}_{h_1,h_2}$ is the matrix arising from playing the skeleton when the hands of the players are set to h_1 and h_2, rescaled by the probability of the pair of hands, $\pi(h_1, h_2)$ specified earlier. The main goal of this section is to show that as the pair of hands (h_1, h_2) varies, the blocks $\boldsymbol{A}_{h_1,h_2}$ exhibit very little variability. That regular structure will then enable us to express the payoff matrix $\boldsymbol{A}$ of Player 1 as a sum of Kronecker products of suitable matrices.

Fix any pair of hands $(h_1, h_2) \in \mathcal{H}_1 \times \mathcal{H}_2$. The block $\boldsymbol{A}_{h_1,h_2}$ of the river endgame's payoff matrix for Player 1 tabulates the payoffs corresponding to the terminal states that can be reached when the players are dealt hands h_1, h_2 (a stochastic events that occurs with probability $\pi(h_1, h_2)$, as defined above). Since the mechanics of the betting round do not depend on the choice of hands, those terminal states are exactly the same terminal states that can be reached in the skeleton of the river endgame. So, the block $\boldsymbol{A}_{h_1,h_2}$ can be written as $\boldsymbol{A}_{h_1,h_2} = \pi(h_1, h_2)\boldsymbol{A}^{\text{skel}}_{h_1,h_2}$, where $\boldsymbol{A}^{\text{skel}}_{h_1,h_2}$ is the payoff matrix induced by the skeleton when the players' hands are set to h_1 and h_2. Furthermore, by separating the contributions $\boldsymbol{F}$ and $\boldsymbol{S}_{h_1,h_2}$ from fold and showdown terminal states respectively, the payoff matrix $\boldsymbol{A}^{\text{skel}}_{h_1,h_2}$ can be written as $\boldsymbol{A}^{\text{skel}}_{h_1,h_2} = \boldsymbol{F} + \boldsymbol{S}_{h_1,h_2}$. The matrix $\boldsymbol{F}$ of payoffs associated to the fold terminal states is straightforward to compute. The initial stacks and pot contributions are known, so the stacks and pot contributions of the players at each node of the skeleton can be easily determined by following the path of (betting) actions from the root of the skeleton to that state (see Figure 1 (Right), and the appendix for a worked out example). We now turn our attention to the matrix of showdown payoffs $\boldsymbol{S}_{h_1,h_2}$. When the Player 1's hand beats the opponent's, the payoff of the player at each showdown terminal state is equal to Player 2's pot contribution—which, by the rules of poker, is equal to Player 1's pot contribution. When the player's hand loses the opponent's, the payoff at each showdown terminal state is the negative amount of the player's pot contribution. Finally, when the hands tie (or are incompatible given the board), the payoffs are all zero. So, introducing the quantity $\gamma(h_1, h_2)$ defined as 1 when hand h_1 beats h_2 (given the board B that was dealt), -1 when hand h_1 is beaten by hand h_2, and 0 when the hands tie or are incompatible given the board, we can write $\boldsymbol{S}_{h_1,h_2} = \gamma(h_1, h_2)\boldsymbol{S}$, where $\boldsymbol{S}$ is the matrix of Player 1's pot contributions at each of showdown terminal states of the skeleton. So, putting all the observations together, we have that $\boldsymbol{A}_{h_1,h_2} = \pi(h_1, h_2)\boldsymbol{F} + \pi(h_1, h_2)\gamma(h_1, h_2)\boldsymbol{S}$ for all hand pairs $(h_1, h_2) \in \mathcal{H}_1 \times \mathcal{H}_2$, and we are ready to formalize the Kronecker-product structure of river endgames in formal terms.

Proposition 1. *Consider a river endgame with board B and hand beliefs μ_1, μ_2 with normalization constant β, and let $\boldsymbol{F}$ and $\boldsymbol{S}$ be the matrices of fold payoffs and showdown payoffs as described above. Introduce the vectors $\boldsymbol{\lambda}_i$ and diagonal matrices $\boldsymbol{\Lambda}_i$ for each player $i = 1, 2$, whose entries are indexed over hands and are defined as $\boldsymbol{\lambda}_i[h_i] = \boldsymbol{\Lambda}_i[h_i, h_i] := \frac{\mu_i(h_i)}{\sqrt{\beta}} \quad \forall i \in \{1, 2\}, h_i \in \mathcal{H}_i$. Furthermore, introduce the $|\mathcal{H}_1| \times |\mathcal{H}_2|$ matrices $\boldsymbol{H}^{\times}, \boldsymbol{W}$, and $\boldsymbol{C}$, defined as*

$$\boldsymbol{W}[h_1, h_2] := \gamma(h_1, h_2), \quad \boldsymbol{C} := \boldsymbol{\mu}_1\boldsymbol{\mu}_2^{\top} - \boldsymbol{\Lambda}_1\boldsymbol{H}^{\times}\boldsymbol{\Lambda}_2,$$

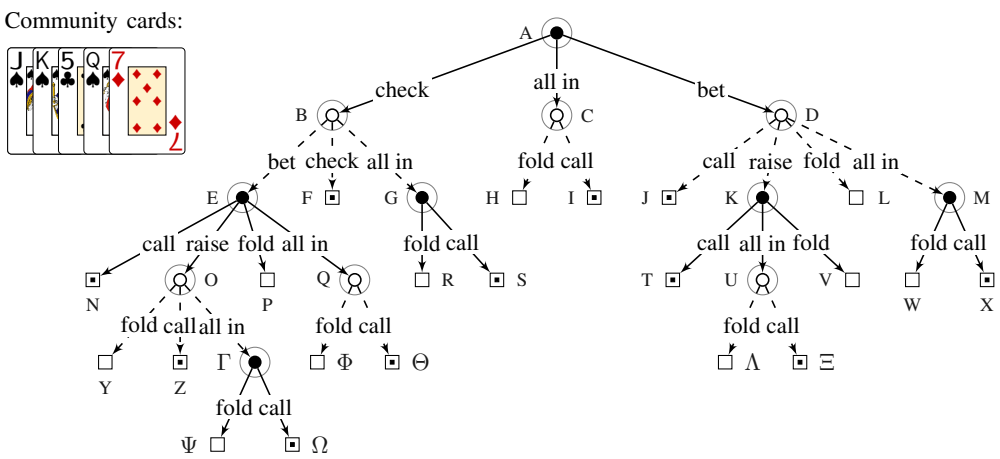
$$\boldsymbol{H}^{\times}[h_1, h_2] := \begin{cases} 1 & \textit{if } h_1, h_2, \textit{ and } B \textit{ are incompatible} \\ 0 & \textit{otherwise}. \end{cases}$$

Then, the payoff matrix $\boldsymbol{A}$ for Player 1 can be written as the sum of Kronecker products

$$\boldsymbol{A} = \boldsymbol{C} \otimes \boldsymbol{F} + (\boldsymbol{\Lambda}_1\boldsymbol{W}\boldsymbol{\Lambda}_2) \otimes \boldsymbol{S}. \tag{3}$$

The ideas presented so far were presented in the context of a river endgame, but they apply directly also to the endgame that starts right after the turn card has been revealed, and more broadly in the full game tree of poker. For the turn endgame, we would start from the skeleton of the first betting round. Only two outcomes are possible: either the game ends in a fold, or the betting round terminates in a non-fold terminal state z, at which point the final card (aka. river card) is revealed and a river endgame begins. Note that because the river card is public, the payoff matrix of the turn endgame is made of diagonal blocks, with each block representing a

[1] Usually, the belief distributions reflect the posterior that each player has over the hands of the opponent, given what they have observed about the opponent's play prior to the river endgame. Here, we make no assumption on how the belief distributions have been formed, and simply take the two distributions as given.



Pot contributions ($)		
F	1,875.0	1,875.0
H	20,000.0	1,875.0
I	20,000.0	20,000.0
J	4,687.5	4,687.5
L	4,687.5	1,875.0
N	4,687.5	4,687.5
P	1,875.0	4,687.5
R	1,875.0	20,000.0
S	20,000.0	20,000.0
T	11,718.8	11,718.8
V	4,687.5	11,718.8
W	4,687.5	20,000.0
X	20,000.0	20,000.0
Y	11,718.8	4,687.5
Z	11,718.8	11,718.8
Φ	20,000.0	4,687.5
Θ	20,000.0	20,000.0
Λ	20,000.0	11,718.8
Ξ	20,000.0	20,000.0
Ψ	11,718.8	20,000.0
Ω	20,000.0	20,000.0

Figure 1: Skeleton of a river endgame. The initial pot contributions for the endgame are $1875 for both players. Each player has a stack size worth $18125 they can play. Every 'bet' and 'raise' action corresponds to first matching the other player's contribution to the pot, and then increasing the player's contribution to the pot by an amount equal to $3/4$ of the cumulative amount in the pot. Black nodes belong to the small blind player, white nodes to the big blind player. The symbol ⊡ denotes a showdown, while □ denotes that one player folded.

river endgame. Because of the diagonal structure, each river endgame can be independently decomposed as in Proposition 1 and sparsified using the techniques we will develop in the next section. This line of reasoning can be composed for each of betting rounds in the game. That shows that Proposition 1 in fact captures the essence of the combinatorial, Kronecker-structure nature of poker games.

4 Sparsification Techniques

We propose two sparsification techniques that directly leverage the Kronecker-product structure of the payoff matrix that we described in Section 3. We will do so with reference to the same symbols used in Proposition 1. We will find the following property of the Kronecker product useful.

Property 1 (Mixed-product rule)**.** *Let* $\boldsymbol{P} \in \mathbb{R}^{m\times n}, \boldsymbol{Q} \in \mathbb{R}^{r\times s}, \boldsymbol{C} \in \mathbb{R}^{n\times \ell}, \boldsymbol{D} \in \mathbb{R}^{s\times q}$ *be arbitrary matrices. Then,* $(\boldsymbol{PC}) \otimes (\boldsymbol{QD}) = (\boldsymbol{P} \otimes \boldsymbol{Q})(\boldsymbol{C} \otimes \boldsymbol{D})$.

The two techniques we propose operate on expression (3) by sparsifying its two terms $\boldsymbol{C} \otimes \boldsymbol{F}$ and $(\boldsymbol{\Lambda}_1 \boldsymbol{W} \boldsymbol{\Lambda}_2) \otimes \boldsymbol{S}$ separately by fundamentally using the mixed-product rule for Kronecker products. Both techniques sparsify the term $\boldsymbol{C} \otimes \boldsymbol{F}$ using the same strategy:

$$\begin{aligned}\boldsymbol{C} \otimes \boldsymbol{F} &= (\boldsymbol{\mu}_1 \boldsymbol{\mu}_2^\top - \boldsymbol{\Lambda}_1 \boldsymbol{H}^\times \boldsymbol{\Lambda}_2) \otimes \boldsymbol{F} \\ &= -(\boldsymbol{\Lambda}_1 \boldsymbol{H}^\times \boldsymbol{\Lambda}_2) \otimes \boldsymbol{F} + (\boldsymbol{\mu}_1 \boldsymbol{\mu}_2^\top) \otimes (\boldsymbol{IF}) \\ &= -(\boldsymbol{\Lambda}_1 \boldsymbol{H}^\times \boldsymbol{\Lambda}_2) \otimes \boldsymbol{F} + (\boldsymbol{\mu}_1 \otimes \boldsymbol{I})(\boldsymbol{\mu}_2 \otimes \boldsymbol{F}^\top)^\top, \qquad (4)\end{aligned}$$

where we used the bilinearity of Kronecker products in the second equality, and the mixed product rule in the last one. The two techniques differ in the way they handle the term $(\boldsymbol{\Lambda}_1 \boldsymbol{W} \boldsymbol{\Lambda}_2) \otimes \boldsymbol{S}$ in (3).

4.1 Technique A

The first technique sparsifies the term $(\boldsymbol{\Lambda}_1 \boldsymbol{W} \boldsymbol{\Lambda}_2) \otimes \boldsymbol{S}$ by recursively sparsifying the 'win-lose' matrix $\boldsymbol{W}$. Specifically, it first computes a sparsification $\boldsymbol{W} = \hat{\boldsymbol{W}} + \boldsymbol{U}_W \boldsymbol{V}_W^\top$ (in our experiments, we do so by using the general heuristic described in (Zhang and Sandholm 2020)), and then uses the mixed-product rule of Kronecker product to write (all steps are in the appendix)

$$\begin{aligned}(\boldsymbol{\Lambda}_1 \boldsymbol{W} \boldsymbol{\Lambda}_2) \otimes \boldsymbol{S} &= (\boldsymbol{\Lambda}_1 \hat{\boldsymbol{W}} \boldsymbol{\Lambda}_2) \otimes \boldsymbol{S} \\ &\quad + \Big((\boldsymbol{\Lambda}_1 \boldsymbol{U}_W) \otimes \boldsymbol{I}\Big)\Big((\boldsymbol{\Lambda}_2 \boldsymbol{V}_W) \otimes \boldsymbol{S}^\top\Big)^\top, \qquad (5)\end{aligned}$$

where the equality follows from the mixed-product rule. Putting (4) and (5) together, we obtain:

Proposition 2. *The payoff matrix* (3) *admits the sparsification* $\boldsymbol{A} = \hat{\boldsymbol{A}} + \boldsymbol{U}\boldsymbol{M}^{-1}\boldsymbol{V}^\top$*, where*

$$\begin{aligned}\hat{\boldsymbol{A}} &:= (\boldsymbol{\Lambda}_1 \hat{\boldsymbol{W}} \boldsymbol{\Lambda}_2) \otimes \boldsymbol{S} - (\boldsymbol{\Lambda}_1 \boldsymbol{H}^\times \boldsymbol{\Lambda}_2) \otimes \boldsymbol{F}, \\ \boldsymbol{U} &:= \Big[(\boldsymbol{\Lambda}_1 \boldsymbol{U}_W) \otimes \boldsymbol{I} \;\Big|\; \boldsymbol{\mu}_1 \otimes \boldsymbol{I}\Big], \qquad \boldsymbol{M} := \boldsymbol{I}, \\ \boldsymbol{V} &:= \Big[(\boldsymbol{\Lambda}_2 \boldsymbol{V}_W) \otimes \boldsymbol{S}^\top \;\Big|\; \boldsymbol{\mu}_2 \otimes \boldsymbol{F}^\top\Big].\end{aligned}$$

4.2 Technique B

The second technique leverages the fact that the hands of each player can be ranked by their strength. When that is done (ignore for now incompatible hands) each row of the win-lose matrix $\boldsymbol{W}$ begins with zero or more columns equal to -1, followed by zero or more columns with value 0, followed by zero or more columns with value 1. As the hand of the row player becomes stronger, the number of -1's on the row decreases, and the number of 1's increases. Hence, the matrix obtained by subtracting from each row of $\boldsymbol{W}$ the previous line must be very sparse. We can compactly represent the operation of subtracting from each row of $\boldsymbol{W}$ the preceding row via the matrix operation $\boldsymbol{Y} := \boldsymbol{DW}$, where the lower bidiagonal matrix $\boldsymbol{D}$ has value 1 on the main diagonal, and value -1 in the diagonal below the main diagonal.

Then,

$$(\mathbf{\Lambda}_1 \boldsymbol{W} \mathbf{\Lambda}_2) \otimes \boldsymbol{S} = \Big(\mathbf{\Lambda}_1 (\boldsymbol{D}^{-1}\boldsymbol{Y}) \mathbf{\Lambda}_2\Big) \otimes (\boldsymbol{I}\boldsymbol{S})$$
$$= (\mathbf{\Lambda}_1 \otimes \boldsymbol{I})(\boldsymbol{D} \otimes \boldsymbol{I})^{-1} \Big((\mathbf{\Lambda}_2 \boldsymbol{Y}^\top) \otimes \boldsymbol{S}^\top\Big)^\top, \quad (6)$$

and we can state the following result.

Proposition 3. *The payoff matrix* (3) *admits the sparsification* $\boldsymbol{A} = \hat{\boldsymbol{A}} + \boldsymbol{U}\boldsymbol{M}^{-1}\boldsymbol{V}^\top$*, where*

$$\hat{\boldsymbol{A}} := -(\mathbf{\Lambda}_1 \boldsymbol{H}^\times \mathbf{\Lambda}_2) \otimes \boldsymbol{F},$$
$$\boldsymbol{U} := \Big[\mathbf{\Lambda}_1 \otimes \boldsymbol{I} \;\Big|\; \boldsymbol{\mu}_1 \otimes \boldsymbol{I}\Big], \qquad \boldsymbol{M} := \left[\begin{array}{c|c} \boldsymbol{D} \otimes \boldsymbol{I} & \\ \hline & \boldsymbol{I} \end{array}\right],$$
$$\boldsymbol{V} := \Big[(\mathbf{\Lambda}_2 \boldsymbol{Y}^\top) \otimes \boldsymbol{S}^\top \;\Big|\; \boldsymbol{\mu}_2 \otimes \boldsymbol{F}^\top\Big].$$

4.3 Postprocessing

After computing any payoff matrix sparsification, we further slightly decrease its size by removing columns from $\boldsymbol{V}$ that are identically zero. This process is perhaps best exemplified in the case of Technique A, where $\boldsymbol{M} = \boldsymbol{I}$. Suppose that the j-th column of $\boldsymbol{V}$ is zero. Then, given any vector $\boldsymbol{x}$, the j-th row of the vector $\boldsymbol{V}^\top\boldsymbol{x}$ will be zero. Hence, we can safely discard the j-th column of $\boldsymbol{U}$, potentially decreasing the size of the sparsification.

When $\boldsymbol{M}$ is not the identity (as is the case for Technique B), the process is only slightly more involved. In the rest of the discussion, we will assume that $\boldsymbol{M}$ is lower triangular, and that all the entries on its main diagonal are equal to 1. Suppose that the j-th column of $\boldsymbol{V}$ is identically zero. Then, the j-th row of $\boldsymbol{V}^\top\boldsymbol{x}$ is zero, for any vector $\boldsymbol{x}$. We can take advantage of that fact when computing $\boldsymbol{M}^{-1}\boldsymbol{V}^\top\boldsymbol{x}$, that is, when solving the system $\boldsymbol{M}\boldsymbol{y} = \boldsymbol{V}^\top\boldsymbol{X}$. In particular, the j-th row of the system is of the form $\boldsymbol{y}_j + \sum_{i<j} a_i \boldsymbol{y}_i = 0$, which implies that $\boldsymbol{y}_j = -\sum_{i<j} a_i \boldsymbol{y}_i$. Hence, the j-th entry of $\boldsymbol{y}$ is a linear combination of other rows of $\boldsymbol{y}$ and does not need to be stored explicitly. In other words, we can remove any reference to $\boldsymbol{y}_j$ from the system, and replace it with $-\sum_{i<j} a_i \boldsymbol{y}_i$. In the case of Technique B, that operation is especially cheap, given that each row of $\boldsymbol{M}$ always has at most two nonzeros (so, $\boldsymbol{y}_j$ is simply substituted with $\boldsymbol{y}_i$ for some $i < j$). Because $\boldsymbol{y}_j$ is treated implicitly as a linear combination of other entries in $\boldsymbol{y} = \boldsymbol{M}^{-1}\boldsymbol{V}^\top\boldsymbol{x}$ we can simply adjust $\boldsymbol{U}$ by removing the j-th column from the matrix, and sum it, multiplied by a_i, to column i.

5 Experimental Results

We experimentally compare the sparsification techniques introduced in Section 4 on eight River endgames that were actually played in the *Brains vs AI* competition where superhuman performance was reached by an AI, *Libratus*, against four top specialist professional players in no-limit Texas hold'em in January 2017. Unless otherwise indicated, each endgame uses the betting abstraction used by *Libratus* (a description is available in the appendix), which contains significantly more bet sizes than the simple betting abstraction used in the sparsification experiments of Zhang and Sandholm (2020); so, we are addressing significantly larger games. In all games, isomorphic hands (Gilpin and Sandholm 2007; Johanson et al. 2011; Waugh 2013) were collapsed into a single meta-hand, as is standard in computational experiments on poker. All experiments were conducted on a computer with 32GB of RAM and an Intel CPU with 16 (virtual) cores, each with a nominal speed of 2.40GHz.

5.1 Computing the Sparsification

We compare the sparsifications computed by the two techniques we introduced in Section 4 against the general sparsification technique of Zhang and Sandholm (2020). We compare both the time to compute the sparsification and the size (*i.e.*, number of nonzeros) of the resulting sparsification. We ran the iterative algorithm of Zhang and Sandholm (2020) with a fixed random seed and a cap on the number of sparsifying iterations set to 1000. We reused the same implementation of the general technique of Zhang and Sandholm in the implementation of our Technique A to sparsify matrix $\boldsymbol{W}$. We implemented all algorithms in C++, using the Eigen library to provide the implementation of linear algebraic objects such as sparse matrices and vectors.[2] Full results are in available in Table 1.

The algorithm by Zhang and Sandholm could scale up to river endgame 4 (a game with 220 million terminal states) before running out of memory. Our techniques could handle all eight endgames. In terms of sparsity, the technique by Zhang and Sandholm is able to consistently reduce the number of nonzeros required to represent the payoff matrix by a factor in the range 20-50. Our Technique A increases sparsity by a factor between 100 and 200. Our Technique B increases sparsity by a factor between 200 and 400, producing sparsifications that are consistently roughly twice as small as Technique A. In terms of time required by the sparsification algorithm to compute the sparsification in memory, the algorithm by Zhang and Sandholm requires an amount of time in the order of hours, whereas our techniques require between 300 milliseconds and 1 second to compute the sparsification by directly leveraging the Kronecker structure of the endgame. In summary, our techniques consistently produce dramatically better sparsifications while at the same time requiring orders of magnitude less compute time to generate.

5.2 Computation of an Optimal Basis for Nash Equilibrium

In this subsection we show that our sparsification techniques enable—to our knowledge for the first time in the large endgames we test on—the computation of a Nash equilibrium strategy that is an optimal basic (i.e., vertex) solution

[2] Zhang and Sandholm (2020) recommend using a custom implementation for implicit matrices to enhance performance. Judging from the results in their paper, that modification would not change our evaluation. For example, when using a small betting abstraction, they report that their optimized implementation took 68s seconds on river endgame 7 (the smallest game we test on). Our techniques, on the significantly larger betting abstraction we test on, took less than 500ms for the same game.

Game	Unsparsified size	Zhang & Sandholm		Technique A		Technique B	
		Size	Time	Size	Time	Size	Time
River 7	5.09×10^7	1.74×10^6	12m 31s	4.07×10^5	420ms	2.74×10^5	318ms
River 6	6.03×10^7	1.97×10^6	15m 00s	3.64×10^5	580ms	2.70×10^5	454ms
River 8	9.59×10^7	4.15×10^6	34m 12s	5.63×10^5	594ms	3.93×10^5	436ms
River 2	1.77×10^8	1.10×10^7	2h 21m	1.02×10^6	748ms	6.72×10^5	567ms
River 4	2.21×10^8	1.10×10^7	2h 30m	1.26×10^6	480ms	7.76×10^5	624ms
River 1	4.47×10^8	oom	oom	2.27×10^6	889ms	1.60×10^6	699ms
River 3	4.76×10^8	oom	oom	2.76×10^6	1.04s	1.65×10^6	722ms
River 5	4.79×10^8	oom	oom	2.52×10^6	1.02s	1.65×10^6	733ms

Table 1: Comparison between different sparsification techniques. 'oom': out of memory.

to the Nash equilibrium LP (2), as discussed in the first bullet point of Section 2. In our experiments we used the state-of-the-art solver Gurobi to solve the LP. Full results can be found in Table 2, where we measures the time required by Gurobi to solve the LPs, not including the time required to compute the sparsifications (where applicable).

In all games, we solved for a strategy for Player 1. When the LP was left unsparsified, the solver could barely start, immediately running out of memory in River 8. We avoided running experiments with the unsparsified LP beyond River 8. The technique of Zhang and Sandholm (2020) (set up as described in Section 5.1) did not run out of memory, but caused Gurobi to terminate abnormally due to numeric instability in River 8 and River 2. In the games for which the unsparsified LP and the LP sparsified using Zhang and Sandholm's technique could be solved, the performance of Gurobi on the latter was 5x-60x worse than with our sparsification techniques. Using our techniques, we were able to compute—for the first time—an optimal basis for Nash equilibrium (and correspondigly, the exact value of the game) in all eight river endgames. Overall, Technique B outperformed Technique A in the larger games by 1x–3x.

5.3 Computation of a Least-Exploitable Deterministic Strategy

In this subsection we investigate another application that is enabled for the first time by our sparsification technique: the computation of the least-exploitable (that is, strongest against a fully rational agent, aka. minimax) *deterministic* strategy, as described in the second bullet point of Section 2. This application relies on the ability to run linear integer programming technology; in our experiments we used the state-of-the-art solver Gurobi. We investigate computing least-exploitable deterministic strategies in all eight river endgames, using the full *Libratus* betting abstraction in the three smallest games and endgame 4, and the smaller betting abstraction used by Zhang and Sandholm (2020) in the remaining games as Gurobi struggled to solve the larger games with the larger abstraction. We only report data for games sparsified with Technique B, which was found to be the most scalable technique in the previous subsection. In Table 3 we report, for each endgame, an upper bound on the price of determinism that Gurobi was able to certify, and how long it took Gurobi to reach that price of determinism. The experiments show that deterministic strategies are able to extract at least $\approx 98\%$ of the value of in all the games!

5.4 First-Order Methods

As mentioned in the third bullet point of Section 2, our sparsification techniques enable a straightforward parallel method for computing the gradients of the utility function of the game required by first-order methods at each iterations to compute approximate Nash equilibrium strategies. To showcase that application, we implemented a GPU version of

Game	Unsparsif.	ZS20	Techn. A	Techn. B
River 7	8m 51s	2m 38s	⋆ 24.14s	27.09s
River 6	2m 35s	6m 07s	⋆ 6.83s	7.29s
River 8	oom	trouble	⋆ 1m 55s	2m 43s
River 2	—	trouble	38m 34s	⋆ 21m 8s
River 4	—	—	21m 55s	⋆ 17m 18s
River 1	—	—	2h 58m	⋆ 2h 18m
River 3	—	—	3h 54m	⋆ 3h 17m
River 5	—	—	7h 09m	⋆ 2h 34m

Table 2: Computation of an optimal basis for Nash equilibrium. 'oom': out of memory. 'trouble': Gurobi indicated a numeric error in its log.

Game	Time	Price of determinism
River 7	9m 41s	$< 1.16\%$
River 6	25m 29s	$< 1.18\%$
River 8	11m 48s	$< 1.10\%$
River 2†	1h 11m	$< 1.74\%$
River 4	17h 09m	$< 1.42\%$
River 1†	35m 03s	$< 1.00\%$
River 3†	40m 37s	$< 0.94\%$
River 5†	1h 03m	$< 1.60\%$

Table 3: Computation of a least-exploitable deterministic strategy for Player 1, and the corresponding price of determinism.

†Experiments marked with this symbol were conducted on the smaller betting abstraction of Zhang and Sandholm (2020) rather than the original one used by *Libratus*, as Gurobi was far from a good solution after 12 hours on the larger betting abstraction.

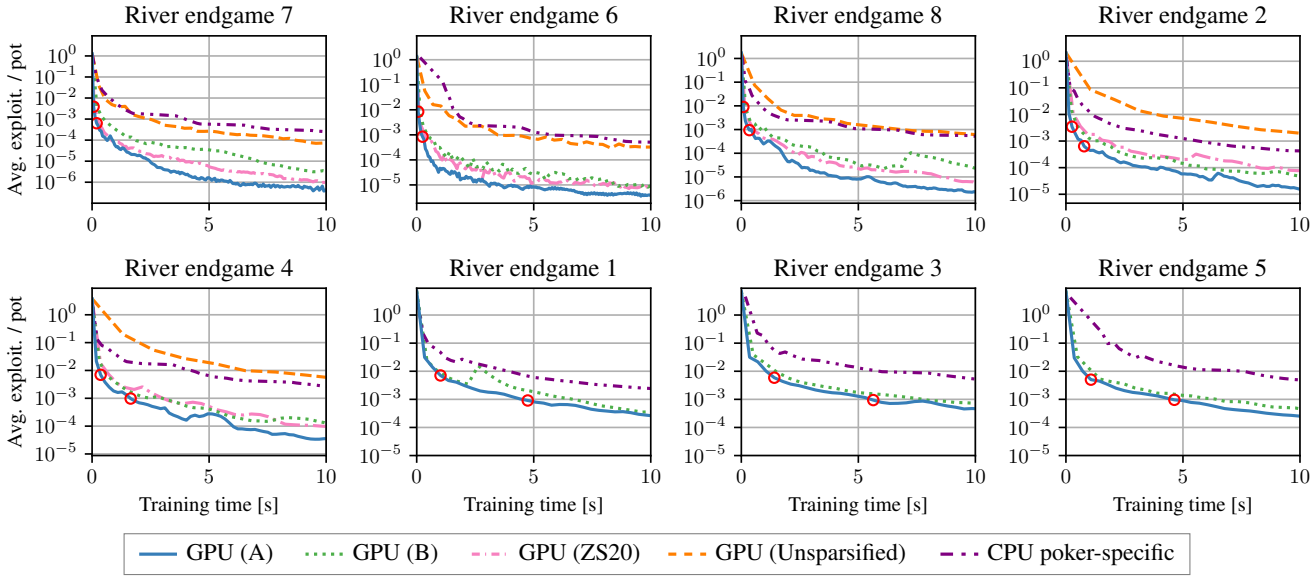


Figure 2: Approximate Nash equilibrium computation using different first-order methods, including our GPU implementation of Discounted CFR leveraging the sparsification techniques. The red circles mark the first time 'GPU (A)' reaches average exploitability less than 1% and 0.1%.

the state-of-the-art CFR variant for poker, *Discounted CFR (DCFR)* (Brown and Sandholm 2019a). Our GPU version of the algorithm was implemented within Nvidia's CUDA framework and run on a laptop-grade Quadro T2000 GPU. It gains from parallelism by updating strategies in parallel for each possible hands of the players. We use the highly-tuned Cusparse libraries to represent, manipulate, and operate on, sparse matrices. We compare two versions of the same code. The first, which we call '*GPU (Unspars.)*' in Figure 2, computes each gradient $\boldsymbol{Ax}$ by explicitly performing the matrix-vector multiplication. The second version leverages the payoff matrix sparsification $\boldsymbol{A} = \hat{\boldsymbol{A}} + \boldsymbol{U}\boldsymbol{M}^{-1}\boldsymbol{V}^{\top}$ to compute the gradient $\boldsymbol{Ax}$ as described in the third bullet of Section 2. Depending on which sparsification is used, we call this version of the GPU implementation '*GPU (ZS20)*' (for Zhang and Sandholm (2020)), '*GPU (A)*' and '*GPU (B)*'. We also compared against a parallel, CPU-based state-of-the-art poker-specific implementation of DCFR that includes the computational shortcuts described by Johanson et al. (2011), denoted '*CPU poker-specific*'. That algorithm had access to all 16 CPU cores. Results are in Figure 2. The y axis measures the average exploitability of the strategy profile within the betting abstraction (equal to half of the Nash saddle point gap), normalized by the total amount of money in the pot at the beginning of the river endgame. Strategies with a relative exploitability of 0.1%-1% are generally considered suitable for play against top human poker professionals. The x axis measures wall-clock time, not including the time to compute the sparsification of the payoff matrix (where applicable). Our Technique A consistently outperforms Technique B, due the absence of the extra operation of solving a triangular system. Our GPU implementation based on Technique A significantly outperforms all other algorithms, and is able to compute strong approximate Nash equilibrium strategies suitable for play against human poker professionals within 5-6 seconds in the worst case, including the time required to compute the sparsification. In many games, it required less than two seconds. These times are well within the norms of usual speed of poker play.

6 Conclusions

We showed that *Kronecker structure* present in games enables the design of specialized payoff matrix sparsification techniques. Those techniques in turn enable optimization algorithms (such as interior-point methods, the simplex method, and integer programming technology) to scale to real-world poker endgames that were previously impossible to handle for those methods, due to the huge size of the payoff matrix of the game. The ability to apply out-of-the-box commercial solvers in games is important, as it enables one to quickly explore questions such as the computation of exact (within numeric tolerance) Nash equilibria, vertex Nash equilibria needed for equilibrium refinements, and least-exploitable deterministic strategies. Furthermore, they significantly speed up parallel first-order game-solving algorithms. We show state-of-the-art speed on a GPU.

Acknowledgments

This material is based on work supported by the National Science Foundation under grants IIS-1718457, IIS-1901403, and CCF-1733556, and the ARO under award W911NF2010081. We thank the anonymous referees for useful comments, and Noam Brown for providing the poker endgames and CPU-based state-of-the-art poker-specific implementation of DCFR that we use in the experiments.

References

Billings, D.; Davidson, A.; Schaeffer, J.; and Szafron, D. 2002. The Challenge of Poker. *Artificial Intelligence*, 134(1-2): 201–240.

Bowling, M.; Burch, N.; Johanson, M.; and Tammelin, O. 2015. Heads-up Limit Hold'em Poker is Solved. *Science*, 347(6218).

Brown, N.; and Sandholm, T. 2017a. Reduced Space and Faster Convergence in Imperfect-Information Games via Pruning. In *International Conference on Machine Learning (ICML)*.

Brown, N.; and Sandholm, T. 2017b. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, eaao1733.

Brown, N.; and Sandholm, T. 2019a. Solving imperfect-information games via discounted regret minimization. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Brown, N.; and Sandholm, T. 2019b. Superhuman AI for multiplayer poker. *Science*, 365(6456): 885–890.

Chen, B.; and Ankenman, J. 2006. *The Mathematics of Poker*. ConJelCo.

Davis, T.; Waugh, K.; and Bowling, M. 2019. Solving large extensive-form games with strategy constraints. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Farina, G.; Gatti, N.; and Sandholm, T. 2018. Practical Exact Algorithm for Trembling-Hand Equilibrium Refinements in Games. In *Conference on Neural Information Processing Systems (NIPS)*.

Farina, G.; Kroer, C.; and Sandholm, T. 2021a. Better Regularization for Sequential Decision Spaces: Fast Convergence Rates for Nash, Correlated, and Team Equilibria. In *ACM Conference on Economics and Computation (EC)*.

Farina, G.; Kroer, C.; and Sandholm, T. 2021b. Faster Game Solving via Predictive Blackwell Approachability: Connecting Regret Matching and Mirror Descent. In *AAAI Conference on Artificial Intelligence*.

Ganzfried, S.; and Sandholm, T. 2010. Computing equilibria by incorporating qualitative models. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Gilpin, A.; and Sandholm, T. 2007. Lossless Abstraction of Imperfect Information Games. *Journal of the ACM*, 54(5).

Hoda, S.; Gilpin, A.; Peña, J.; and Sandholm, T. 2010. Smoothing Techniques for Computing Nash Equilibria of Sequential Games. *Mathematics of Operations Research*, 35(2).

Johanson, M.; Waugh, K.; Bowling, M.; and Zinkevich, M. 2011. Accelerating Best Response Calculation in Large Extensive Games. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Koller, D.; Megiddo, N.; and von Stengel, B. 1996. Efficient Computation of Equilibria for Extensive Two-Person Games. *Games and Economic Behavior*, 14(2).

Kroer, C. 2019. First-Order Methods with Increasing Iterate Averaging for Solving Saddle-Point Problems. *arXiv preprint arXiv:1903.10646*.

Kroer, C.; Farina, G.; and Sandholm, T. 2018. Solving Large Sequential Games with the Excessive Gap Technique. In *Neural Information Processing Systems (NIPS)*.

Kuhn, H. W. 1950. A Simplified Two-Person Poker. In Kuhn, H. W.; and Tucker, A. W., eds., *Contributions to the Theory of Games*, volume 1 of *Annals of Mathematics Studies, 24*, 97–103. Princeton, New Jersey: Princeton University Press.

Moravčík, M.; Schmid, M.; Burch, N.; Lisý, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; and Bowling, M. 2017. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*.

Morrill, D.; D'Orazio, R.; Lanctot, M.; Wright, J. R.; Bowling, M.; and Greenwald, A. 2021. Efficient Deviation Types and Learning for Hindsight Rationality in Extensive-Form Games. In *International Conference on Machine Learning (ICML)*.

Nash, J. 1950a. Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences*, 36: 48–49.

Nash, J. 1950b. *Non-cooperative games*. Ph.D. thesis, Princeton University.

Nemirovski, A. 2004. Prox-method with rate of convergence O(1/t) for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1).

Nesterov, Y. 2005. Excessive Gap Technique in Nonsmooth Convex Minimization. *SIAM Journal of Optimization*, 16(1).

Romanovskii, I. 1962. Reduction of a Game with Complete Memory to a Matrix Game. *Soviet Mathematics*, 3.

Tammelin, O. 2014. Solving large imperfect information games using CFR+. *arXiv preprint arXiv:1407.5042*.

von Stengel, B. 1996. Efficient Computation of Behavior Strategies. *Games and Economic Behavior*, 14(2): 220–246.

Waterman, D. 1970. Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence*, 1(1): 121–170.

Waugh, K. 2013. A fast and optimal hand isomorphism algorithm. In *AAAI Workshop on Computer Poker and Incomplete Information*.

Zadeh, N. 1977. Computation of Optimal Poker Strategies. *Operations Research*, 25(4): 541–562.

Zhang, B.; and Sandholm, T. 2020. Sparsified linear programming for zero-sum equilibrium finding. In *International Conference on Machine Learning*, 11256–11267. PMLR.

Zinkevich, M.; Bowling, M.; Johanson, M.; and Piccione, C. 2007. Regret Minimization in Games with Incomplete Information. In *Neural Information Processing Systems (NIPS)*.