Sparsified Linear Programming for Zero-Sum Equilibrium Finding

Brian Hu Zhang 1  Tuomas Sandholm 1 2 3 4

Abstract
Computational equilibrium finding in large zero-sum extensive-form imperfect-information games has led to significant recent AI breakthroughs. The fastest algorithms for the problem are new forms of counterfactual regret minimization (Brown & Sandholm, 2019). In this paper we present a totally different approach to the problem, which is competitive and often orders of magnitude better than the prior state of the art. The equilibrium-finding problem can be formulated as a linear program (LP) (Koller et al., 1994), but solving it as an LP has not been scalable due to the memory requirements of LP solvers, which can often be quadratically worse than CFR-based algorithms. We give an efficient practical algorithm that factors a large payoff matrix into a product of two matrices that are typically dramatically sparser. This allows us to express the equilibrium-finding problem as a linear program with size only a logarithmic factor worse than CFR, and thus allows linear program solvers to run on such games. With experiments on poker endgames, we demonstrate in practice, for the first time, that modern linear program solvers are competitive against even game-specific modern variants of CFR in solving large extensive-form games, and can be used to compute exact solutions unlike iterative algorithms like CFR.

1. Introduction
Imperfect-information games model strategic interactions between agents that do not have perfect knowledge of their current situation, such as auctions, negotiations, and recreational games such as poker and battleship. Linear programming (LP) can be used to solve—that is, to find a Nash equilibrium in—imperfect-information two-player zero-sum perfect-recall games (Koller et al., 1994). However, due mostly to memory usage (see e.g., Zinkevich et al., 2007 or Brown & Sandholm, 2019), it has generally been thought of as impractical for solving large games. Thus, a series of other techniques has been developed for solving such games. Most prominent among these is the counterfactual regret minimization (CFR) family of algorithms (Zinkevich et al., 2007). These algorithms work by iteratively improving both player’s strategies until their time averages converge to an equilibrium. They have a worst-case bound of $O(1/\varepsilon^2)$ iterations needed to reach accuracy $\varepsilon$, and more recent improvements, most notably CFR+ (Tammelin, 2014) and discounted CFR (DCFR) (Brown & Sandholm, 2019) mean that algorithms from the CFR family remain the state of the art in practice for solving large games, and have been used as an important part of the computational pipelines to achieve superhuman performance in benchmark cases such as heads-up limit (Bowling et al., 2015) and no-limit (Brown & Sandholm, 2017) Texas hold’em.

Several families of algorithms have theoretically faster convergence rates than those of the CFR family. First-order methods (Hoda et al., 2010; Kroer et al., 2015) have a theoretically better convergence guarantee of $O(1/\varepsilon)$ (or even $\log(1/\varepsilon)$ with a condition number (Gilpin et al., 2012)), but in practice perform worse than the newest algorithms in the CFR family (Kroer et al., 2018; Brown & Sandholm, 2019). Standard algorithms for LP are known that converge at rate $O(\log(1/\varepsilon))$, but for the most part, these algorithms require storage of the whole payoff matrix explicitly, which the CFR family does not, and often require superlinear time per iteration (with respect to the number of nonzero entries of the LP constraint matrix), which is prohibitive when the game is extremely large.

In this paper we investigate how to reduce the space requirements of LP solvers by factoring the possibly dense payoff matrix of an extensive-form game. A long body of work investigates the problem of decomposing, or factoring, a given matrix $A$ as the product of other matrices, with some objective in mind. Studied objectives include the speedup of certain operations, as in the LU or Cholesky factorizations, and approximation of the matrix $A$ in a certain norm, as in the singular value decomposition (SVD). Our objective in

---

1Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
2Strategic Machine, Inc.
3Strategy Robot, Inc.
4Optimized Markets, Inc.. Correspondence to: Brian Hu Zhang <bhzhang@cs.cmu.edu>, Tuomas Sandholm <sandholm@cs.cmu.edu>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).
this work is sparsity: we investigate the problem of factoring a matrix $A$ into the product of two matrices $U$ and $V$ that are much sparser than $A$. This differs from the usual low-rank approximation in that the optimization objective is different (0-norm, that is, number of non-zero entries, instead of the 2-norm, that is, the square root of sum of squares), and that the matrices $U$ and $V$ might not be low rank (and in fact in poker they will high rank that is linear in the number of sequences in the game).

We are not aware of any prior application-independent work that addresses this problem. The SVD approximates $A$ in the wrong norm for this purpose: 2-norm approximations will in the general case have fully dense residual, which is not desirable. The body of work on sparse PCA (e.g., Zou & Xue, 2018) focuses on low-rank sparse factorizations. That still mostly focuses on 2-norm approximations, and the runtime of the algorithm usually scales with the rank of the factorization as well as the size of $A$. In our cases, an optimal factorization may have high or even full rank (and yet be sparse), and our matrices are large enough that quadratic (or worse) dependence on $\|A\|_0$, which is often seen in these algorithms, is unacceptable. Our goal is to find such a factorization efficiently. Neyshabur & Panigrahy (2013) address the same problem, but restrict their attention to matrices that are known a priori to be the product of sparse matrices with entries drawn independently from a nice distribution. This is not the case in our setting. Richard et al. (2014) attack a related but still substantially different problem, of finding a sparse factorization when we know a priori a good bound on the sparsity of each row or column of the factors. This, too, is not true in our setting: no such bound may even exist, much less be known.

Our main technical contribution is a novel practical matrix factorization algorithm that greatly reduces the size of the game payoff matrix in many cases. This matrix factorization allows LP algorithms to run in far less memory and time than previously known, bringing the memory requirement close to that of CFR. We demonstrate in practice that this method can reduce the size of a payoff matrix by multiple orders of magnitude, yielding improvements in both the time and space efficiency of solving the resulting LP. This makes our approach—automated matrix sparsification followed by LP—superior to domain-independent versions of the fastest CFR variant. If high accuracy is desired, our domain-independent approach in many cases outperforms even a highly customized poker-specific implementation of the fastest CFR variant (Brown & Sandholm, 2019).

We show experiments with the primal simplex, dual simplex, and the barrier method as the LP solver. The barrier method runs in polynomial time but each iteration is heavy in terms of memory and time. For that reason, we present techniques that significantly speed up a recent $O(\log^2(1/\epsilon))$ LP algorithm (Yen et al., 2015) that has iteration time and memory linear in the number of nonzero entries in the constraint matrix, and show experiments with that as well. Our experiments show interesting performance differences among the LP solvers as well.

2. Preliminaries

**Extensive-form games.** We study the standard representation of games which can include sequential and simultaneous moves, as well as imperfect information, called an extensive-form game. It consists of the following. (1) A set of players $\mathcal{P}$, usually identified with positive integers. Random chance, or “nature” is also considered a player, and will be referred to as player 0. (2) A finite tree $H$ of histories or nodes, rooted at some initial state $\emptyset \in H$. Each node is labeled with the player (possibly nature) who acts at that node. The set of leaves, or terminal states, in $H$ will be denoted $Z$. The edges connecting any node $h \in H$ to its children are labeled with actions. (3) For each player $i \in \mathcal{P}$, a utility function $u_i : Z \rightarrow \mathbb{R}$. (4) For each player $i \in \mathcal{P}$, a partition of the nodes at which player $i$ acts into a collection $I_i$ of information sets. In each information set $I \in I_i$, every pair of nodes $h, h' \in I$ must have the same set of actions. (5) For each node $h$ at which nature acts, a distribution $\sigma_0(h)$ over the actions available at that node.

For any history $h \in H$ and any player $i \in \mathcal{P}$, the sequence $h[i]$ of player $i$ at node $h$ is the sequence of information sets reached and actions played by player $i$ on the path from the root node to $h$. The set of sequences for player $i$ is denoted $S_i$. A player $i$ has perfect recall if $h[i] = h'[i]$ whenever $h$ and $h'$ are in the same information set $I \in I_i$. In this work, we will focus our attention on two-player zero-sum games of perfect recall; i.e., games in which $\mathcal{P} = \{1, 2\}$, $u_1 = -u_2$, and both players have perfect recall. For simplicity of notation, the opponent of player $i$ will be denoted $-i$.

A behavior strategy (hereafter simply strategy) $\sigma_i$ for player $i$ is, for each information set $I \in I_i$ at which player $i$ acts, a distribution $\sigma_i(I)$ over the actions available at that infoset. When an agent reaches information set $I$, it chooses an action according to $\sigma_i(I)$. A pair $(\sigma_1, \sigma_2)$ of behavior strategies, one for each player, is a strategy profile. The reach probability $\pi^*(h)$ is the probability that node $h$ will be reached, assuming that player $i$ plays according to strategy $\sigma_i$, and all other players (including nature) always choose actions leading to $h$ when possible. This definition extends to sets of nodes or to sequences by summing the reach probabilities.

The best response value $BRV(\sigma_{-i})$ for player $i$ against an opponent strategy $\sigma_{-i}$ is the largest achievable value; i.e., in a two-player game, $BRV(\sigma_{-i}) = \max_{\sigma_i} u_i(\sigma_i, \sigma_{-i})$. A strategy $\sigma_i$ is an $\varepsilon$-best response to opponent strategy $\sigma_{-i}$ if
We first give a brief overview of the algorithm. We are interested in LPs of the standard form (2) and their duals.

**Sparse linear programming.** Yen et al. (2015) give a generic algorithm for solving LPs in the standard form (2) and their duals where \( A \in \mathbb{R}^{m \times n} \). Consider the convex subproblem

\[
\min_{y_{LP} \geq 0} b^T_{LP} y_{LP} \quad \text{s.t.} \quad -A_{LP}^T y_{LP} \leq c_{LP}
\]

for some given initial solution \( \hat{y} \in \mathbb{R}^m \) and real number \( \eta > 0 \). The dual of this subproblem is

\[
\min_{x,z} c_{LP}^T x_{LP} + \frac{\eta}{2} \left\| A_{LP} x_{LP} - b_{LP} + z_{LP} + \frac{1}{\eta} \hat{y} \right\|_2^2
\]

\[
\text{s.t.} \quad x_{LP} \geq 0, z_{LP} \geq 0
\]

(3)

The approach is shown in Algorithm 1. In Line 2, the solution to Problem (3) is computed via either a randomized coordinate descent (RC) or a projected Newton-CG (PG) algorithm; the details are not important here. The breakthrough of Yen et al. (2015) is an implementation of these inner loops in \( O(\|A_{LP}\|_0) \) time, rather than \( O(mn) \) or worse. At each iteration, \( x^* \) is infeasible in the original problem since the quadratic regularization term in (3) does not push slightly infeasible solutions much at all. \( y^* \) is infeasible since \((x^*, z^*) \) is a suboptimal solution to (3). Thus, Algorithm 1 works with infeasible solutions to the LP, which must be projected back into the feasible space.

**Algorithm 1** Augmented Lagrangian algorithm for solving linear programs (Yen et al., 2015)

**Input:** initial dual solution guess \( \hat{y} \in \mathbb{R}^m \), parameter \( \eta > 0 \)

**Output:** primal-dual solution pair \((x^*, \hat{y})\)

1: **loop**

2: \( \text{let } (x^*, z^*) \text{ be an approximate solution to Problem (3) given the current } \hat{y} \text{ and } \eta. \)

3: \( \text{set } \hat{y} \leftarrow \hat{y} + \eta(\hat{y}_{LP} - b_{LP} + z^*) \)

4: **if** necessary (as detailed by Yen et al. (2015)), **increase** \( \eta \) by a constant factor

**Theorem 1** (Theorem 3 in Yen et al., 2015). After \( O((\log(1/\varepsilon)) \text{ outer iterations of Algorithm 1, each of which is run for } O(\log(1/\varepsilon)) \text{ inner iterations, we have } d(\hat{y}, S) \leq \varepsilon \text{ where } S \subseteq \mathbb{R}^m \text{ is the set of dual-optimal solutions and } d \text{ is Euclidean distance}. \)

The \( O \) in the above theorem hides problem-dependent constants such as condition numbers. This theoretical guarantee applies to the dual solution, and not the primal. Thus, to find a primal-dual solution pair, in theory we must run Algorithm 1 twice: on the primal (to find a dual solution) and then the dual (to find a primal solution). In practice, the primal solution from the first run already has extremely low exploitability, so the second run would be unnecessary.

The rest of the paper covers our new contributions.

3. **Adapting the** \( O(\log^2(1/\varepsilon)) \) **Sparse LP Solver**

In order to make the above LP algorithm fast for game solving, we had to make a modification and also deal with the caveat of eternally infeasible solutions \( x_{LP} \) and \( y_{LP} \).
3.1. Limiting the Number of Inner Iterations

Yen et al. (2015) give an implementation of their algorithm, which they call \( LP_{\text{sparse}} \). In it, the inner loop runs until either (1) it converges to a sufficiently small error tolerance, or (2) some prescribed iteration limit is hit. The iteration limit is set to increase exponentially every time it is hit. In practice, we found this to be far too aggressive, leading to inner loops that take prohibitively long (an hour or more on two-player no-limit Texas hold’em endgames). Thus, we instead we only allow the number of inner iterations to grow linearly with respect to the number of outer iterations. Since both the outer and inner loop lengths are bounded by \( O(\log(1/\varepsilon)) \) in Theorem 1, this does not change the theoretical performance guarantee of the algorithm, and it leads to a significant speedup in practice.

3.2. Normalizing Infeasible Solutions

Algorithm 1 will output an infeasible solution pair. To retrieve a valid behavior strategy (feasible solution), we first project into the positive orthant (i.e., zero out any negative entries), and then normalize each information set in topological order, starting with the root. This results in a strategy pair whose Nash gap we can evaluate. This normalization step roughly maintains the guarantee of Theorem 1:

\textbf{Theorem 2.} Suppose \( x_{\text{LP}} = (x, z) \) is an infeasible solution to (1) such that \( d((x, z), S) \leq \varepsilon \), where \( S \) is the set of optimal solutions to (1). Then the above normalization yields a (feasible) strategy with exploitability at most \( \varepsilon n^4 \|A\|_\infty \), where \( n \) is the total number of sequences between the two players.

A proof is in the appendix. The above bound is loose, but it is unnecessary to improve it for the theoretical punchline: combining Theorems 1 and 2, we see that the LP algorithm converges to a strategy with exploitability \( \varepsilon \) in \( O(\log^2(1/\varepsilon)) \) inner iterations (where the \( O \) possibly hides problem-dependent constants), assuming \( A \) is normalized (i.e., \( \|A\|_\infty \) is fixed to, say, 1).

4. Sparse Factorization

In many games, the payoff matrix \( A \) is somewhat dense. This occurs when the number of terminal game tree nodes, \( |Z| \), is large compared to the total number of sequences \( |S_1| + |S_2| \), that is, when a significant fraction of the sequence pairs represent valid terminal nodes. In most normal-form (a.k.a. matrix-form) games, \( A \) is fully dense, whereas in extensive-form games of perfect information, \( A \) is extremely sparse (because the number of terminal nodes equals the total number of terminal sequences between the players). In most real games, the value of each entry \( A_{ij} \) can be computed in constant time from the indices \( i \) and \( j \) alone based on the rules of the game, with a minimal amount of auxiliary memory, so \( A \) can be stored implicitly. In these cases, the sparse LP solver is at a disadvantage compared to the CFR family of algorithms. CFR can run with only implicit access to \( A \). Its memory usage is thus \( O(|S_1| + |S_2|) \). LP solvers, on the other hand, require a full description of \( A \), which here will have size \( O(|Z|) \). Our idea here is to make LP solvers practical by carefully compressing \( A \) in a way that standard LP solvers can still handle.

This leads to our main idea. If we can write \( A \) approximately as the product of two matrices; that is, \( A = A + UV^T \), such that \( \|U\|_0 + \|V\|_0 + \|A\|_0 \ll \|A\|_0 \), then we can reformulate the LP (1) as

\[
\max_{x \geq 0, z, w} \quad c^T z \quad \text{s.t.} \quad Bx = b, \quad CTz \leq Vw + \hat{A}^T x, \quad U^T x = w
\]

which, in standard form, has \( O(|B|_0 + |C|_0 + |U|_0 + |V|_0 + |A|_0) \) nonzero constraint matrix entries. In this formulation, we demand that not only \( U \) and \( V \) but also the residual \( \hat{A} \) be sparse. Depending on the density of \( A \) and the quality of the factorization \( A + UV^T \), a good factorization could yield a quadratic improvement in both the time and space used by the LP solver.

When \( A \) is low-rank, SVD would provide such a factorization. However, in many cases, \( A \) is not sparse and not well approximated by a low-rank factorization. Further, even when \( A \) is low-rank, it is possible that, for example, \( A - wv^T \) is a dense matrix (where \( wv^T \) is the best rank-1 approximation to \( A \)), which means that the algorithm would take \( \Omega(\text{mn}) \) time and memory per iteration starting from the second outer iteration. We now give examples of matrices \( A \) for which finding a sparse factorization in our style is superior to finding a standard low-rank factorization (SVD), both in speed and resulting sparsity. An additional example can be found in the appendix.

\textbf{Example 1.} Let \( A_1 = uv^T \) be a rank-one matrix, and let \( A \) be \( A_1 \), except its lower-triangular half has been zeroed out. In general, \( A \) will now be full-rank, and the SVD of \( A \) will not be sparse. However, we can express \( A = UV^T \) with \( \|U\|_0 = \|V\|_0 = O(n \log n) \) as follows. Set \( u_0 = u \) except with its right half zeroed out, and set \( v_0 = v \) except with its left half zeroed out. Then \( u_0v_0^T \) matches the upper-right quadrant of the matrix \( A \), as shown in Figure 1. Moreover, \( A - u_0v_0^T \) is block diagonal, where the two blocks have size \( (n/2) \times (n/2) \) and have the same structure as \( A \) itself. Thus, we may recursively factor the two blocks. The vectors \( u_0 \) and \( v_0 \) both have \( n/2 \) nonzero entries, so the total number of nonzero entries in the factorization is expressed by the recurrence \( S(n) = n + 2S(n/2) \), which solves to \( S(n) = O(n \log n) \). The matrices \( U \) and \( V \) will both have \( \Theta(n) \) columns. This example shows up in practice; the payoff matrix of poker endgames is block diagonal, where the blocks have essentially this form.
We thus resort to an algorithm that may not yield the optimal solution but works extremely well in practice. Algorithm 3 reduces (4) to solving the subproblem

$$\arg\min_{v} \| A - uv^T \| \quad (5)$$

for a given matrix $A$ and now a fixed vector $u$ (the other subproblem is analogous by transposing $A$ and flipping the roles of $u$ and $v$). Again, when $\|\cdot\|$ is the 2-norm, and the optimizer of (5) is just $v^* = Au$, so that the full algorithm is just standard power iteration algorithm for SVD.

**Algorithm 3 Approximating $\arg\min_{u,v} \| A - uv^T \|$**

**Input:** matrix $A \in \mathbb{R}^{m \times n}$

**Output:** vectors $u$, $v$

1: make an initial guess for $u$
2: loop
3: $v \leftarrow \arg\min_{u} \| A - uv^T \|$
4: $u \leftarrow \arg\min_{u} \| A - uv^T \|$

When $\|\cdot\|$ is instead the 0-norm, as seen in Algorithm 4, the optimizer of (5) is the vector $v$ whose $j$th element is the mode of $A_{ij} / u_i$ over all $i$ for which $u_i \neq 0$. Since the objective function (4) cannot increase during the alternating minimization, and the objective values are integral, Algorithm 3 terminates in finitely many iterations at a local optimum. Algorithm 2 is an anytime algorithm. In practice, we terminate it when the number of unsuccessful iterations (the number of iterations in which the condition on line 4 is false) exceeds the number of successful iterations.

**Algorithm 4 Sparse matrix factorization subproblem**

**Input:** matrix $A \in \mathbb{R}^{m \times n}$, vector $u \in \mathbb{R}^{m}$

**Output:** vector $v$ minimizing $\| A - uv^T \|_0$

1: $q \leftarrow$ map from indices to lists of real numbers
2: for each $i$ for which $u_i \neq 0$ do
3: for each nonzero entry $A_{ij}$ in row $i$ of $A$ do
4: append $A_{ij} / u_i$ to $q[j]$
5: $v \leftarrow 0$
6: for each $j$ for which $q[j]$ is nonempty do
7: $M \leftarrow$ mode($q[j]$)
8: count $\leftarrow$ number of times $M$ appears in $q[j]$
9: if count $> \| u \|_0 - \text{len}(q[j])$ then $v_j \leftarrow M$

Algorithms 2-4 constitute an approximate algorithm for sparse matrix factorization. It is not exact for two reasons. First, Algorithm 2 is greedy: at each step of the loop, it chooses the immediate best rank-1 matrix and greedily appends it to $U$ and $V$. This is not always optimal, and in fact in the worst case can already doom the algorithm to have a trivial approximation factor $\Theta(n)$. Second, Algorithm 3 is not exact when $\|\cdot\|$ is the 0-norm or 1-norm, as...
expected from our Theorem 3. Nevertheless, the method works remarkably well as we will show experimentally.\footnote{We also experimented with using the 1-norm as a convex relaxation of the 0-norm. Here, the exact solution to (5) is given by Meng & Xu (2012). This seemed to make no difference in practice, so in the experiments we use the 0-norm.}

5.1. Initialization

For the SVD, the initial guess for \( u \) in Algorithm 3 is usually chosen to point in a random direction (i.e., \( u_i \sim N(0,1) \) are drawn i.i.d.). In our case, this does not work: if we draw \( u \) that way, then as long as each column of \( A \) has at least two nonzero entries, the mode computation in Algorithm 4 will return \( v = 0 \) with probability 1, since \( A_{ij}/u_i \) will be different for each \( i \) with probability 1. This causes the subroutine of Algorithm 2 to degenerate, leading to a trivial output. This is troubling for us, since in basically all extensive-form practice, so in the experiments we use the 0-norm. This seemed to make no difference in practice since we do not need to distinguish when \( A \) has been transposed in Algorithm 4.

As stated above, the run time of the algorithm is dominated by the matrix multiplication \( U_i V^T \), which must be performed for every \( i \) where \( u_i \neq 0 \) on the current iteration. On the \( r \)th outer iteration of the algorithm, \( U \) and \( V \) will have \( r \) columns each; therefore, \( V^T \in \mathbb{R}^{r \times n} \), so the matrix multiplication takes time \( O(rn) \). We need to perform \( \|u\|_0 \) of these per inner iteration. Thus, if the algorithm runs for a total of \( R \) outer loop iterations each of which takes \( t \) inner-loop iterations, it will take time

\[
O \left( t \sum_{r=1}^R (\|u_r\|_0 + \|v_r\|_0)rn \right).
\]

In practice, the number of inner iterations \( t \) per outer iteration is usually very small, say, 3. As an example, if the algorithm correctly factors a matrix of the structure in Example 1, the \( r \)th outer loop iteration will find a block of size \( O(1/r) \times O(1/r) \). Thus each inner loop iteration just takes time \( O(n) \), and there will be \( O(n) \) iterations, so that the whole algorithm runs in time \( O(n^2) = O(\|A\|_0) \).

In most extensive-form games, the payoff matrix \( A \) is block diagonal. In this case, running the factorization algorithm on \( A \) is equivalent to running it on each of the blocks individually, and has the same run time as running the algorithm on each block separately. Indeed, if \( u \) is initialized to a random basis vector \( e_i \), the algorithm’s entire run, and all its operations—including the critical matrix multiplication \( U_i V^T \)—will not escape the block to which row \( i \) of matrix \( A \) belongs. Thus, for example, running the algorithm on a matrix with blocks of size \( k \times k \), each of which has the structure of Example 1, will still take time \( O(\|A\|_0) \).

5.2. Implementation with Implicit Matrices

A major problem with the above algorithm is that a straightforward implementation of it would store and modify the matrix \( A \) in order to factor it. In the setting we are considering, \( A \) is often too big to store in memory: the number of nonzero entries in \( A \) may be several orders of magnitude greater than the number of rows or columns. In these settings, we would like to be able to implement the algorithm with only implicit access to \( A \). Formally, we assume access to \( A \) via only an immutable oracle that, given an index \( i \), retrieves the list of nonzero entries, and their indices, in the \( i \)th row or \( i \)th column of \( A \).

The immutability of \( A \) is the biggest roadblock here. Several changes need to be made to Algorithms 2-4 to accommodate this. First, Line 7 of Algorithm 2 is no longer possible. Thus, Line 3 of Algorithm 2, must be revised to read \( \arg\min_{u,v} \| A - UV^T - uv^T \| \), and the matrices \( U \) and \( V \) must be passed through to Algorithms 3 and 4. On Line 3 of Algorithm 4, querying the \( i \)th row of \( A - UV^T \) requires a matrix multiplication \( U_i V^T \), where \( U_i \) is the \( i \)th row of \( U \).

5.3. Run-time Analysis

The run time of the algorithm depends heavily on the structure of \( A \). The worst case run time is \( O(\|A\|_0 n^2) \), since every inner iteration runs in at most quadratic time and removes at least one nonzero entry from \( A \). In practice it runs dramatically faster than that, and we will now present a rough analysis, valid in most typical cases. For simplic-
Table 1. Experiments on explicitly specified games. \( \text{Gap} \) is the target Nash gap to which \( \text{LPsparse'} \) and CFR were run. \( \text{fnnz} \) is the total number of nonzero elements that resulted from running our matrix factorization algorithm, reported only when the factorization algorithm had nontrivial effect. \( \text{Simplex} \), \( \text{Barrier} \), \( \text{LPsparse'} \), and CFR are the wall-clock times, in seconds, that those four algorithms took to achieve the desired Nash gap. All times greater than 2 hours (7200 seconds) are estimated via linear regression on the log-log convergence plot. Gurobi was time-limited to half an hour (1800 seconds) because each game had at least one method that solved the game well within this limit. Since it is difficult to estimate the convergence rate of Gurobi’s solver, Gurobi timeouts are simply indicated with a (T).

| Game                          | Gap |
|-------------------------------|-----|
| 9-card Leduc poker            | .0001 |
| 13-card Leduc poker           | .0001 |
| 5x2 battleship m=2 n=1        | .0001 |
| 4x3 battleship m=2 n=1        | .0001 |
| 3x2 battleship m=4 n=1        | .0001 |
| 3x2 battleship m=3 n=2        | .0001 |
| sheriff N=10000 B=100         | .0001 |
| sheriff N=100 B=1000          | .0001 |
| 5x5 battleship m=3 n=2        | .0001 |
| 4x4 battleship m=4 n=1        | .0001 |
| 4x4 battleship m=3 n=2        | .0001 |
| 4-rank goofspiel              | 1.74 |
| 5-rank goofspiel              | 1.74 |
| NLH river endgame A           | 0.0684 |
| NLH river endgame B           | 0.0178 |

The implicit implementation of our factorization algorithm (Section 5.2) allows us to scale our method to larger games than previously possible. We hence ran experiments testing this implementation on heads-up no-limit Texas Hold’em poker endgames encountered by the superhuman agent Liberatus (Brown & Sandholm, 2017). To align with Brown & Sandholm (2019), we used a simple action abstraction: the bets are half-pot, full-pot, and all-in, and the raises are full-pot and all-in. All results are expressed in terms of the standard metric, namely big blinds (bb). The starting stacks are 200 big blinds per player as in the Libratus match against humans. We tested on eight real river endgames (i.e., endgames that start on the fourth betting round) and a single manually-generated small turn endgame (i.e., endgames that start on the third betting round) where the pot already has half of the players’ wealth, so only a single additional bet or raise is possible.
In this experiment we used an optimized poker-specific C++ implementation of DCFR. This implementation includes optimizations such as those of Johanson et al. (2011), which shave an $O(k)$ factor off the runtime of CFR, where in the case of Texas hold’em poker, $k = 1326$ is the number of possible hands a player may have, and Brown & Sandholm (2015), which prune game lines that are dynamically determined not to be part of the optimal solution. For the LP solver, we use Gurobi’s simplex and barrier methods. Both primal and dual simplex were run, and only the better of the two results is shown in Table 2. We also tested Gurobi without the factorization algorithm. In this case, we do not include results for the barrier method, because it timed out or ran out of memory in all the cases. All algorithms were again restricted to a single core. DCFR was run for the amount of time taken by the fastest LP variant that used factoring. For example, if Gurobi took 200 seconds to solve a game, and the factorization algorithm took 100 seconds, CFR was run for 300 seconds. The results are in Table 2 and representative convergence plots showing anytime performance are in the appendix.

The factorization algorithm reduced the size of the game by a factor of 52–80 and the resulting payoff matrix had density (i.e., nonzeros divided by rows plus columns) 7.8–9.5. This is expected: poker payoff matrices are block diagonal, where the blocks are $k \times k$ and rank one with the lower-triangular half negated. Thus, they basically have the structure of Example 1, in which we saw a compression from density $k \approx 2^{10}$ to density $\log k \approx 10$, which is exactly the compression we are seeing here.

The DCFR implementation we tested against is especially optimized to solve no-limit turn endgames. It thus may have some inefficiencies when handling river endgames. We estimate that these inefficiencies lose a factor of approximately 20 in time and space on river endgames relative to a river-optimized implementation. However, importantly, these inefficiencies pale in comparison to the speedups gained by game-specific poker speedups (e.g., Johanson et al., 2011), which save a factor of approximately $k = 1326$ in time (but not space). This strongly suggests that our method would be significantly faster than any non-game-specific implementation of CFR or any modern variant. Furthermore, the memory usage of simplex, after factorization, is only a factor of $\log k \approx 10$ worse than the game-specific CFR (which stores the constraint matrix implicitly) in the case of all these endgames, which means it is often practical to use...
LP solvers even on extremely large games with dense payoff matrices, as long as the constraint matrix is factorable.

Since primal simplex and dual simplex give respectively only primal-feasible and dual-feasible solutions, anytime performance of simplex is measured by running both simultaneously, and measuring the Nash gap between the current primal and dual solutions at each time checkpoint, using Gurobi’s reported objective values. While Gurobi does not allow retrieval of these anytime solutions when its presolver is turned on, in principle they can be retrieved easily using the presolver’s mapping, which unfortunately Gurobi does not expose to the end user. The convergence plots in Figure 2 show roughly what we would expect. CFR has a very stable convergence curve (until it hits too high precision, at which point numerical stability issues start kicking in, and the convergence plot looks weird). The LP solvers start out slow (especially due to the sometimes nontrivial time requirements of the factorization algorithm) but catch up with and often eventually exceed the performance of DCFR, before again very often stopping due to numerical issues. Even on turn endgames, LP algorithms consistently outperform a hypothetical non-game-specific implementation of DCFR—which we define to be 500 times slower than the poker-specific DCFR—due to the additional factor of \( k \approx 1326 \) in the density of the payoff matrix, and hence the additional cost of the gradient computation in DCFR.

7. Conclusion and Future Research

We presented a matrix factorization algorithm that yields significant reduction in sparsity. We showed how the factored matrix can be used in an LP to solve zero-sum games. This reduces both the time and space needed by LP solvers. On explicitly represented games, this significantly outperforms the prior state-of-the-art algorithm, DCFR. It also made LP solvers competitive on large games that are implicitly defined by a compact set of rules—even against an optimized game-specific DCFR implementation. There are many interesting directions for future research, such as (1) further improving the factorization algorithm, (2) investigating the explicit form of an optimal factorization in special cases, and (3) parallelizing the factorization algorithm.
Acknowledgements

This material is based on work supported by the National Science Foundation under grants IIS-1718457, IIS-1617590, IIS-1901403, and CCF-1733556, and the ARO under awards W911NF1710082 and W911NF2010081.

References

Bowling, M., Burch, N., Johanson, M., and Tammelin, O. Heads-up limit hold’em poker is solved. Science, 347 (6218), January 2015.

Brown, N. and Sandholm, T. Regret-based pruning in extensive-form games. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), 2015.

Brown, N. and Sandholm, T. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. Science, pp. eaao1733, Dec. 2017. Print version 359(6374):418–424, 2018.

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

Farina, G., Ling, C. K., Fang, F., and Sandholm, T. Correlation in extensive-form games: Saddle-point formulation and benchmarks. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), 2019.

Gillis, N. and Vavasis, S. A. On the complexity of robust PCA and \ell_1\text{-norm low-rank matrix approximation.} Mathematics of Operations Research, 43(4):1072–1084, 2018.

Gilpin, A., Peña, J., and Sandholm, T. First-order algorithm with \(O(\ln(1/\epsilon))\) convergence for \(\epsilon\)-equilibrium in two-person zero-sum games. Mathematical Programming, 133(1–2):279–298, 2012. Conference version appeared in AAAI-08.

Golub, G. H. and Van Loan, C. F. Matrix Computations. Johns Hopkins University Press, 1996.

Gurobi Optimization, L. Gurobi optimizer reference manual, 2019.

Hoda, S., Gilpin, A., Peña, J., and Sandholm, T. Smoothing techniques for computing Nash equilibria of sequential games. Mathematics of Operations Research, 35(2), 2010.

Johanson, M., Waugh, K., Bowling, M., and Zinkevich, M. Accelerating best response calculation in large extensive games. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2011.

Koller, D., Megiddo, N., and von Stengel, B. Fast algorithms for finding randomized strategies in game trees. In Proceedings of the 26th ACM Symposium on Theory of Computing (STOC), 1994.

Kroer, C., Waugh, K., Kilinc-Karzan, F., and Sandholm, T. Faster first-order methods for extensive-form game solving. In Proceedings of the ACM Conference on Economics and Computation (EC), 2015.

Kroer, C., Farina, G., and Sandholm, T. Solving large sequential games with the excessive gap technique. In Conference on Neural Information Processing Systems (NIPS), 2018.

Meng, D. and Xu, Z. Divide-and-conquer method for L1 norm matrix factorization in the presence of outliers and missing data. arXiv, abs/1202.5844, 2012.

Neyshabur, B. and Panigrahy, R. Sparse matrix factorization. arXiv, abs/1311.3315, 2013.

Richard, E., Obozinski, G., and Vert, J. Tight convex relaxations for sparse matrix factorization. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), 2014.

Southey, F., Bowling, M., Larson, B., Piccione, C., Burch, N., Billings, D., and Rayner, C. Bayes’ bluff: Opponent modelling in poker. In Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI), July 2005.

Tammelin, O. Solving large imperfect information games using CFR+. CoRR, abs/1407.5042, 2014.

von Stengel, B. Efficient computation of behavior strategies. Games and Economic Behavior, 14(2):220–246, 1996.

Yen, I. E., Zhong, K., Hsieh, C., Ravikumar, P., and Dhillon, I. S. Sparse linear programming via primal and dual augmented coordinate descent. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), pp. 2368–2376, 2015.

Zinkevich, M., Bowling, M., Johanson, M., and Piccione, C. Regret minimization in games with incomplete information. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS), 2007.

Zou, H. and Xue, L. A selective overview of sparse principal component analysis. Proceedings of the IEEE, 106(8):1311–1320, 2018.
A. Proof of Theorem 2

The key to the proof is to bound how much this naive normalization changes the point \( x \). Let \((x^*, z^*)\) be the result of projecting \((x, z)\) into the optimal set \( S \).

**Lemma.** Let \( x' \) be the result of normalizing \( x \) according to the given scheme, and \( i \) be an information set at depth \( d \) (with the root defined to be at depth 0). Then we have \(|x'_i - x^*_i| \leq \varepsilon d \sqrt{n}\).

**Proof.** By induction on the sequence-form strategy tree for player \( x \), starting at the root. At the root node \( i = 0 \), the claim is clearly true because \( x_0 = 1 \) in any feasible solution \( x \). Now consider any information set with parent \( x_{i_0} \) and children \( x_i := (x_{i_1}, \ldots, x_{i_k}) \) at depth \( d \). From the theorem statement, we have \(|x_i - x^*_i| \leq \varepsilon \), and since \( x^* \) is feasible, we have \( \sum_{j=1}^k x^*_{i_k} = x^*_{i_0} \). It follows that

\[
\left| \sum_{j=1}^k x_{i_k} - x^*_{i_0} \right| \leq \sum_{j=1}^k |x_{i_k} - x^*_{i_k}| + |x^*_{i_0} - x^*_{i_0}| \leq \varepsilon \sqrt{k} + \varepsilon (d - 1) \sqrt{n} \leq \varepsilon d \sqrt{n}
\]

by triangle inequality and inductive hypothesis, and noting that \( k \leq n \). But the normalization acts by picking \( x'_i \) so that \( \sum_{j=1}^k x'_{i_k} = x'_{i_0} \), and it moves all the \( x_{i_k} \)s in the same direction; thus, each one can move by at most \( \varepsilon d \sqrt{n} \), completing the induction. \( \square \)

With this lemma in hand, we now prove the theorem.

**Proof of Theorem.** Since \( d \leq n \) (each depth must have at least one information set), it follows from the lemma that \(|x' - x^*|_2 \leq \varepsilon n^2 \). But the best response function \( \min_y x^T A y \) (with feasibility constraints on \( y \)) is a pointwise minimum of Lipschitz functions \( x^T v \) for each \( v = A y \) and \( y \) feasible, hence itself Lipschitz, with Lipschitz constant

\[
\max_{y} \|A y\|_2 \leq \max_{y} \|A y\|_1 \leq \|A\|_1 \max_{y} \|y\|_\infty = \|A\|_1 \leq n^2 \|A\|_\infty,
\]

where \( \|A\|_1 \) is the sum of the magnitudes of the nonzero entries of \( A \). The desired theorem follows. \( \square \)

B. Another Example of the Utility of Sparse Factorization

**Example 3.** Let \( A \) be the \( n \times (n + 1) \) matrix given by \( A = [I_n \ 0] + [0 \ I_n] \), where \( I_n \) is the \( n \times n \) identity, and 0 is a column vector of zeros. So, \( A \) is the matrix whose \((i, j)\) entry is 1 exactly when \( j = i \) or \( j = i + 1 \). By direct computation, the SVD of this matrix is \( A = U \Sigma V^T \) where \( U \) and \( V \) are **fully dense**, and the SVD is unique (in the usual sense, that is, up to signs and permutations) since all the singular values are. Thus, taking an SVD would have the result of **increasing** the number of nonzeros from \( 2n \) to \( \Theta(n^2) \), which is the opposite of what we want. Thus, although in this case there will not be a good sparse factorization, using SVD make the problem worse.

C. Benchmark Games in Experiment 1

We tested on the following benchmark games from the literature:

- **Leduc poker** (Southey et al., 2005) is a small variant of poker, played with one hole card and three community cards.
- **Battleship** (Farina et al., 2019) is the classic targeting game, with two parameters: \( m \) is the number of moves (shots) a player may take, and \( n \) is the number of ships on the board. All ships have length 2. A player scores a point only for sinking a full ship.
• Sheriff (Farina et al., 2019) is a simplified Sheriff of Nottingham game, modified to be zero-sum, played between a smuggler and a sheriff. The smuggler selects a bribe amount $b \in [0, B]$ and a number of illegal items $n \in [0, N]$ to try to smuggle past the sheriff. The sheriff then decides whether to inspect. If the sheriff does not inspect the cargo, then the smuggler scores $n - b$. If the sheriff inspects and finds no illegal items ($n = 0$), then the smuggler scores 3. If the sheriff inspects, and $n > 0$, then the smuggler scores $-2n$. The smuggler has far more sequences than the sheriff in this game.

• No-limit hold-em (NLH) river endgames are endgames encountered by the poker-playing agent Libratus (Brown & Sandholm, 2017), using the action abstraction used by Libratus. They both begin on the last betting round, when all five community cards are known. The normalization of $\|A\|_\infty = 1$ means that in these endgames, a Nash gap of 1 corresponds to 0.075 big blinds. Due to the explicit storage of the payoff matrix in this experiment, only extremely small no-limit endgames can be tested. In particular, endgame A here is the same as endgame 7 in the next experiment (with a finer abstraction), and endgame B is the same as endgame A except with the starting pot size doubled to make the game smaller.