Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence

Alpha-Beta Pruning for Games with Simultaneous Moves

Abdallah Saffidine
LAMSADE, Université Paris-Dauphine, 75775 Paris Cedex 16, France
Email: abdallah.saffidine@dauphine.fr

Hilmar Finnsson
Reykjavík University, Menntavegi 1, 101 Reykjavík, Iceland
Email: hif@ru.is

Michael Buro
University of Alberta, Edmonton, T6G 2E8, Canada
Email: mburo@ualberta.ca

Abstract

Alpha-Beta pruning is one of the most powerful and fundamental MiniMax search improvements. It was designed for sequential two-player zero-sum perfect information games. In this paper we introduce an Alpha-Beta-like sound pruning method for the more general class of "stacked matrix games" that allow for simultaneous moves by both players. This is accomplished by maintaining upper and lower bounds for achievable payoffs in states with simultaneous actions and dominated action pruning based on the feasibility of certain linear programs. Empirical data shows considerable savings in terms of expanded nodes compared to naive depth-first move computation without pruning.

1 Introduction

When searching game trees, especially in a competitive setting, significant benefits can be achieved by pruning branches which under no circumstances can affect the decision being made at the root.

The best known pruning method is the Alpha-Beta algorithm (Knuth and Moore 1975; Russell and Norvig 2010). It applies to sequential zero-sum two-player games with perfect information such as Chess and Checkers. Alpha-Beta maintains upper and lower value bounds to decide whether branches can be cut. This type of pruning can lead to considerable search reductions — essentially doubling the search depth over the original MiniMax algorithm when given the same search time.

After its discovery, sound Alpha-Beta style pruning has been extended to other game types and game tree search algorithms. E.g., for sequential two-player zero-sum games with perfect information and chance nodes, *-MiniMax search safely prunes irrelevant subtrees (Ballard 1983), and (Sturtevant and Korf 2000; Sturtevant 2005) describe Alpha-Beta like pruning rules for general-sum games and games with more than two players. Recently, Monte Carlo Tree Search (MCTS), which is a type of simulation-based best-first search algorithm, has been extended to allow for Alpha-Beta style pruning (Cazenave and Saffidine 2011).

In this paper we generalize Alpha-Beta pruning to two-player zero-sum games with simultaneous moves. These games are a subset of the multi-agent environment (MAE) as described in (Schiffel and Thielscher 2010) which we will use as a reference in order to place our contribution within the world of studied game types.

The paper is structured as follows: First, we give the necessary technical background before introducing Simultaneous Move Alpha-Beta (SMAB) pruning — and explaining it in detail. We then describe how SMAB pruning can be used in the context of depth-first search and present empirical data to show its effectiveness. We conclude the paper with ideas for future work in this area.

2 Background

2.1 MiniMax and Alpha-Beta

The MiniMax value of a game tree is calculated based on the assumption that the two players, called *Max* and *Min*, will choose their next move such that when it is *Max*'s turn he will select the action that maximizes his gain while *Min* will select the one that minimizes it on his turn. MiniMax values are propagated from the leaves of the game tree to its root using this rule. Alpha-beta utilizes the MiniMax value to prune a subtree when it has proof that a move will not affect the decision at the root node. This happens when a partial search of the subtree reveals that the opponent has the opportunity to lower an already established MiniMax value backed up from a different subtree.

2.2 Score Bounded MCTS

An MCTS solver which backs up exact MiniMax values of the sequential zero-sum two-outcome game *Lines of Action* was introduced in (Winands, Björnsson, and Saito 2008). Score bounded MCTS (Cazenave and Saffidine 2011) expands on this idea and generalized the MCTS solver concept to any sequential zero-sum game. Score bounded search allows for pruning in the absence of exact MiniMax values as long as there is some information available to establish bounds.

Because simulations do not usually methodically explore the game tree, it is to be expected that we cannot easily as-

Table 1: Pruning in Multi-Agent Environments

Sequential	Zero-sum	Agents	Pruning
Yes	Yes	Two	$\alpha\beta$
Yes	Yes	Any	(Sturtevant and Korf 2000)
Yes	No	-	(Sturtevant 2005)
No	Yes	Two	This paper

sign MiniMax values to the states when we explore them as we are only sampling the subtree below. Even though we may not have explored every reachable state, the sampling information builds up and can be used to get tighter and tighter bounds on state values. These bounds are called pessimistic and optimistic, referring to the payoff *Max* believes he can get in the worst and best case, respectively. The default bounds are the minimum and maximum achievable values. Instead of backing up a MiniMax value, the bounds of a state are deduced from the bounds of subsequent states and used in Alpha-Beta fashion by checking whether lower and upper bounds coincide.

2.3 Multi-Agent Environment (MAE)

The Multi-Agent Environment (MAE) formally describes discrete and deterministic multi-agent domains (Schiffel and Thielscher 2010). It can be seen as a transition system where each node corresponds to a state and transitions are associated with joint actions executed by the participating agents. It is useful to classify MAEs along several dimensions:

Definition 1. *An MAE is* single-player *if the number of agents is exactly one,* two-player *if the number of agents is exactly two, and* multiplayer *otherwise.*

Definition 2. *An MAE is* (purely) sequential *if in any state, there is at most one agent with more than one legal action.*

Definition 3. *An MAE is* zero-sum *if the sum of utilities of all agents is constant in all final states.*

Table 1 summarizes related work of where pruning has been achieved in the context of MAE and clarifies where our contribution lies.

2.4 Nash Equilibrium and Normal-Form Games

A Nash equilibrium is a strategy profile for all players for which no player can increase his payoff by deviating unilaterally from his strategy. In the case of zero-sum two-player games, all Nash equilibria result in the same payoff, called the *value* of the game. When faced with simultaneous actions, Nash equilibrium strategies are often mixed strategies in which actions are performed with certain probabilities (e.g. the only Nash equilibrium strategy for ROCK-PAPER-SCISSORS is playing Rock, Paper, and Scissors with probability $1/3$ each).

Two-player zero-sum games are often presented in normal-form which in a matrix lists payoffs for player *Max* for all action — or more generally pure strategy — pairs. Throughout this paper, player *Max* chooses rows, and player *Min* chooses columns. When working with normal-form games it is sometimes possible to simplify them based on action domination. This happens when no matter how the opponent acts, the payoff for some action a is always less or equal to the payoff for some other action b or a mixed strategy not containing a. In this situation there is no incentive to play action a and it can be ignored. The possibility of actions being dominated opens the door for pruning.

Example 1. *Consider game G below. The row player will only select action A_2 if the value of subgame H is greater than 5. Now consider subgame H: no matter what the values of cells (A_3, B_3) and (A_4, B_3) are, the best value the row player can hope for at this point is 4. As a result, we do not even need to compute the exact value for H and it can be pruned.*

$G =$

	B_1
A_1	5
A_2	*value(H)*

$H =$

	B_2	B_3
A_3	4	?
A_4	3	?

2.5 Applicable Game Domains and General Solution Techniques

The type of games our pruning technique applies to can be loosely described as *stacked matrix games*, but they can also be seen as a deterministic non-loopy subclass of recursive games (Everett 1957). This class of games encompasses a small portion of games appearing in the GGP competition such as BIDDING-TICTACTOE. Furthermore, particular instances of this game class have been studied in (Buro 2003; Kovarsky and Buro 2005; Furtak and Buro 2010).

As a subset of general zero-sum imperfect information games, stacked matrix games can be solved by general techniques such as creating a single-matrix game in which individual moves represent pure strategies in the original game. However, because this transformation leads to an exponential blowup, it can only be applied to tiny problems. In their landmark paper, (Koller, Megiddo, and von Stengel 1994) define the sequence form game representation which avoids redundancies present in above game transformation and reduces the game value computation time to polynomial in the game tree size. In the experimental section we present data showing that even for small stacked matrix games, the sequence form approach requires lots of memory and therefore can't solve larger problems. The main reason is that the algorithm doesn't detect the regular information set structure present in stacked matrix games, and also computes mixed strategies for all information sets, which may not be necessary. To overcome these problems (Gilpin and Sandholm 2007) introduce a loss-less abstraction for games with certain regularity constraints and show that Nash equilibria found in the often much smaller game abstractions correspond to ones in the original game. General stacked matrix games don't fall into the game class considered in this paper, but the general idea of pre-processing games to transform them into smaller, equivalent ones may also apply to stacked matrix games.

3 Simultaneous Move Pruning

In this section we assume that, without loss of generality, all payoffs are given in view of row-player *Max*. Moreover, to allow us to use moves as indexes, we assume that there is a

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_{a-1} \\ x_{a+1} \\ \vdots \\ x_m \end{pmatrix}, P = \begin{pmatrix} p_{1,1} & \dots & p_{1,n} \\ \vdots & & \vdots \\ p_{a-1,1} & \dots & p_{a-1,n} \\ p_{a+1,1} & \dots & p_{a+1,n} \\ \vdots & & \vdots \\ p_{m,1} & \dots & p_{m,n} \end{pmatrix}, e = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

$$f = (o_{a,1} \quad \dots \quad o_{a,n})$$

$$x^t P \geq f, 0 \leq x \leq 1, \textstyle\sum_i x_i = 1$$

Figure 1: System of inequalities for deciding whether row action a is dominated. a is dominated and can be pruned if the system of inequalities is feasible.

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_{a-1} \\ x_{a+1} \\ \vdots \\ x_m \end{pmatrix}, p = \begin{pmatrix} p_1 \\ \vdots \\ p_{a-1} \\ p_{a+1} \\ \vdots \\ p_m \end{pmatrix}, e = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

$$x^t p \geq o_a, 0 \leq x \leq 1, \textstyle\sum_i x_i = 1$$

Figure 2: System of inequalities to decide if a row action a can be pruned when there is only one column action.

bijection between legal moves and a subset of consecutive natural numbers starting with 1.

The criterion we use for pruning is similar to that of the original Alpha-Beta algorithm: we prune sub-trees if we have proof that they will under no circumstances improve upon the current guaranteed payoff assuming rational players.

Let q be a position in the game tree with m actions for *Max* and n actions for *Min*. For all $1 \leq i \leq m$ and $1 \leq j \leq n$, we call $q_{i,j}$ the position reached after joint action (i, j) is executed in q. We assume that the information we have gained so far about position $q_{i,j}$ is in the form of a pessimistic bound $p_{i,j}$ and an optimistic bound $o_{i,j}$ on the real value of $q_{i,j}$. For instance, if the value v of $q_{i,j}$ has been determined, we have $p_{i,j} = v = o_{i,j}$. If, however, no information about $q_{i,j}$ is known, we have $p_{i,j} = \text{minval}$ and $o_{i,j} = \text{maxval}$.

To determine if a row action a can be safely pruned from the set of available *Max* actions in the presence of pessimistic payoff bounds $p_{i,j}$ and optimistic payoff bounds $o_{i,j}$ we use linear programming. A sufficient pruning condition is that action a is dominated by a mixed strategy excluding a. Using the given payoff bounds, we need to prove that there is a mixed strategy excluding action a that, when using pessimistic payoff bounds, dominates action a's optimistic payoff bounds. If such a mixed strategy exists then there is no need to consider action a, because a certain mixture of other actions is at least as good.

The system of inequalities (SI) in Figure 1 shows these calculations. If this system is feasible then action a can be pruned. Note that if $n = 1$, i.e. this state features a non-simultaneous action with *Max* to play, the SI reduces to the one shown in Figure 2. This SI is feasible if and only if there exists an action $a' \neq a$ such that $p_{a'} \geq o_a$. This is can be reformulated as pruning action a if $\max p_i \geq o_a$ which matches the pruning criterion in score bounded MCTS (Cazenave and Saffidine 2011) exactly. The analogous SI for pruning dominated column actions is shown in Figure 3.

$$x = (x_1 \quad \dots \quad x_{b-1} \quad x_{b+1} \quad \dots \quad x_m)$$

$$O = \begin{pmatrix} o_{1,1} & \dots & o_{1,b-1} & o_{1,b+1} & \dots & o_{1,n} \\ \vdots & & \vdots & \vdots & & \vdots \\ o_{m,1} & \dots & o_{m,b-1} & o_{m,b+1} & \dots & o_{m,n} \end{pmatrix}$$

$$f = \begin{pmatrix} p_{1,b} \\ \vdots \\ p_{m,b} \end{pmatrix}$$

$$e = (1 \quad \dots \quad 1)$$

$$Ox^t \leq f, 0 \leq x \leq 1, \textstyle\sum_i x_i = 1$$

Figure 3: System of inequalities to decide if a column action b is dominated. b is dominated and can be pruned if the system of inequalities is feasible.

4 Simultaneous Move Alpha-Beta Search

Like the original Alpha-Beta algorithm, we traverse a given game tree in depth-first manner, for each position q using a lower bound α and an upper bound β on the value of q. As soon as we can prove that the value of q lies outside (α, β), we can prune the remaining positions below q and backtrack.

In this section we again assume that payoffs are given in view of row-player *Max* and that for each game state and player we have a bijection between legal moves and move indices starting at 1.

We begin by explaining how to determine the α and β bounds from pessimistic and optimistic value bounds. We then show how this computation can be integrated into a recursive depth-first search algorithm. Finally, we discuss some practical aspects.

4.1 Propagating Bounds

Let q be a position in the game tree and $A = \{1..m\}$ and $B = \{1..n\}$ the move sets for players *Max* and *Min*. For all $(i, j) \in A \times B$, we call $q_{i,j}$ the position reached after joint action (i, j) is executed in q. We assume that the information we have gained so far about position $q_{i,j}$ is in the form of a pessimistic bound $p_{i,j}$ and an optimistic bound $o_{i,j}$ on the real value of $q_{i,j}$. The default bound values are minval and maxval, respectively. Let $q_{a,b}$ be the next position to examine. We are interested in computing $\alpha_{q_{a,b}}$ and $\beta_{q_{a,b}}$ in terms of α, β (the value bounds for q), $p_{i,j}$ and $o_{i,j}$ for $(i, j) \in A \times B$. We first concentrate on computing $\alpha_{q_{a,b}}$, or $\alpha_{a,b}$ for short. $\beta_{a,b}$ can be derived analogously.

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_{a-1} \\ x_{a+1} \\ \vdots \\ x_m \\ x_{m+1} \end{pmatrix}, P = \begin{pmatrix} p_{1,1} & \cdots & p_{1,b-1} & p_{1,b+1} & \cdots & p_{1,n} \\ \vdots & & \vdots & \vdots & & \vdots \\ p_{a-1,1} & \cdots & p_{a-1,b-1} & p_{a-1,b+1} & \cdots & p_{a-1,n} \\ p_{a+1,1} & \cdots & p_{a+1,b-1} & p_{a+1,b+1} & \cdots & p_{a+1,n} \\ \vdots & & \vdots & \vdots & & \vdots \\ p_{m,1} & \cdots & p_{m,b-1} & p_{m,b+1} & \cdots & p_{m,n} \\ \alpha & \cdots & \alpha & \alpha & \cdots & \alpha \end{pmatrix}, e = \begin{pmatrix} p_{1,b} \\ \vdots \\ p_{a-1,b} \\ p_{a+1,b} \\ \vdots \\ p_{m,b} \\ \alpha \end{pmatrix}$$

$$f = (o_{a,1} \quad \cdots \quad o_{a,b-1} \quad o_{a,b+1} \quad \cdots \quad o_{a,n})$$

$\alpha_{a,b} = \max x^t e$, subject to $x^t P \geq f, 0 \leq x \leq 1, \sum_i x_i = 1$, or minval$-1$ if the LP is infeasible

Figure 4: Computing the pessimistic value $\alpha_{a,b}$

$$x = (x_1 \quad \cdots \quad x_{b-1} \quad x_{b+1} \quad \cdots \quad x_n \quad x_{n+1})$$

$$O = \begin{pmatrix} o_{1,1} & \cdots & o_{1,b-1} & o_{1,b+1} & \cdots & o_{1,n} & \beta \\ \vdots & & \vdots & \vdots & & \vdots & \beta \\ o_{a-1,1} & \cdots & o_{a-1,b-1} & o_{a-1,b+1} & \cdots & o_{a-1,n} & \beta \\ o_{a+1,1} & \cdots & o_{a+1,b-1} & o_{a+1,b+1} & \cdots & o_{a+1,n} & \beta \\ \vdots & & \vdots & \vdots & & \vdots & \beta \\ o_{m,1} & \cdots & o_{m,b-1} & o_{m,b+1} & \cdots & o_{m,n} & \beta \end{pmatrix}, f = \begin{pmatrix} p_{1,b} \\ \vdots \\ p_{a-1,b} \\ p_{a+1,b} \\ \vdots \\ p_{m,b} \end{pmatrix}$$

$$e = (o_{a,1} \quad \cdots \quad o_{a,b-1} \quad o_{a,b+1} \quad \cdots \quad o_{a,n} \quad \beta)$$

$\beta_{a,b} = \min ex^t$, subject to $Ox^t \leq f, 0 \leq x^t \leq 1, \sum_i x_i = 1$, or maxval$+1$ if the LP is infeasible

Figure 5: Computing the optimistic value $\beta_{a,b}$

There are two reasons why we might not need to know the exact value of $q_{a,b}$, if it is rather small. Either we have proved that it is so small that a is dominated by a mixed strategy not containing a (shallow pruning), or it is so small that as a result we can prove that the value of q is smaller than α (deep pruning). We can combine both arguments into one LP by adding an artificial action $m+1$ for *Max* that corresponds to *Max* deviating earlier. This action guarantees a score of at least α, i.e. $p_{m+1,j} = \alpha$ for all $j \in B$. We can now restrict ourselves to determining under which condition action a would be dominated by a mixed strategy of actions $M := \{1, \ldots, m+1\} \backslash \{a\}$. To guarantee soundness, we need to look at the situation where a is least expected to be pruned, i.e. when the values of positions $q_{a,j}$ reach their optimistic bounds $o_{a,j}$ and for every other action $i \neq a$, the values of positions $q_{i,j}$ reach their pessimistic bounds $p_{i,j}$.

Consider the set of mixed strategies D dominating a on every column but b, i.e.

$$D = \{x \in I\!R^m_{\geq 0} \mid \sum_i x_i = 1, \forall j \neq b : \sum_{i \in M} x_i p_{i,j} \geq o_{a,j}\}$$

Action a is dominated if and only if a is dominated on column b by a strategy in D. I.e., action a is dominated if and only if value v of $q_{a,b}$ satisfies:

$$\exists x \in D : \sum_{i \in M} x_i p_{i,b} \geq v$$

If D is non-empty, to have the tightest $\alpha_{a,b}$ possible, we maximize over such values:

$$\alpha_{a,b} = \max_{x \in D} \sum_{i \in M} x_i p_{i,b}$$

Otherwise, if D is empty, $q_{a,b}$ can't be bound from below and we set $\alpha_{a,b} =$ minval.

This process can be directly translated into the LP presented in Figure 4. Similarly, the bound $\beta_{q_{a,b}}$ is defined as the objective value of the LP shown in Figure 5.

4.2 Main Algorithm

Algorithm 1 describes how our simultaneous move pruning can be incorporated in a depth-first search algorithm by looping through all joint action pairs first checking trivial exit conditions and if these fail, proceeding with computing optimistic and pessimistic bounds for the entry in questions, and then recursively computing the entry value. We call this procedure Simultaneous Move Alpha-Beta (SMAB) Search.

Theorem: When SMAB is called with q, α, β and $\alpha < \beta$ ...

1. ... it runs in weakly polynomial time in the size of the game tree rooted in q.
2. ... and returns $v \leq \alpha$, then value(q) $\leq v$.
3. ... and returns $v \geq \beta$, then value(q) $\geq v$.
4. ... and returns $\alpha < v < \beta$, then value(q) $= v$.

```
1  SMAB (state q, lower bound α, upper bound β)
2  if q is a terminal state then
3  |  return payoff for q
4  else
5  |  let A = the set of legal moves for the row player;
6  |  let B = the set of legal moves for the col player;
7  |  let p_{i,j} = minval for i ∈ A, j ∈ B;
8  |  let o_{i,j} = maxval for i ∈ A, j ∈ B;
9  |  let P denote the matrix formed by all p_{i,j};
10 |  let O denote the matrix formed by all o_{i,j};
11 |  for each (a, b) ∈ A × B do
12 |  |  if row a and column b not dominated then
13 |  |  |  let α_{a,b} as defined in Fig. 4 restricted to
   |  |  |  non-dominated actions;
14 |  |  |  let β_{a,b} as defined in Fig. 5 restricted to
   |  |  |  non-dominated actions;
15 |  |  |  let q_{a,b} = the state reached after applying
   |  |  |  (a, b) to q;
16 |  |  |  if α_{a,b} ≥ β_{a,b} then
17 |  |  |  |  let v_{a,b} = SMAB (q_{a,b}, α_{a,b}, α_{a,b} + ε) ;
18 |  |  |  |  if v_{a,b} ≤ α_{a,b} then  a is dominated;
19 |  |  |  |  else  b is dominated;
20 |  |  |  else
21 |  |  |  |  let v_{a,b} = SMAB (q_{a,b}, α_{a,b}, β_{a,b}) ;
22 |  |  |  |  if v_{a,b} ≤ α_{a,b} then  a is dominated;
23 |  |  |  |  else if v_{a,b} ≥ β_{a,b} then  b is dominated;
24 |  |  |  |  else  let p_{a,b} = o_{a,b} = v_{a,b};
25 |  |  |  end
26 |  |  end
27 |  end
28 |  return Nash(P restricted to non-dominated actions)
29 end
```

Algorithm 1: Pseudo-code for simultaneous move Alpha-Beta search. Function $\mathrm{Nash}(X)$ computes the Nash equilibrium value of normal-form game payoff matrix X for row player *Max*.

Proof Sketch:

1.: Weakly polynomial run-time in the sub-tree size can be shown by induction on the tree height using the fact that LPs can be solved by interior point methods in weakly polynomial time.
2.,3.,4.: Induction on tree height h. For $h = 0$, SMAB immediately returns the true value. Thus, properties 2.-4. hold. Now we assume they hold for all heights $h \leq k$ and q has height $k + 1$ and proceed with an induction on the number of inner loop iterations claiming that P and O are correctly updated in each step (using the derivations in the previous subsection and the main induction hypothesis) and if line 28 is reached, properties 2.-4. hold. □

4.3 Ordering Move Pairs

Heuristics can be used to initialize $(p_{i,j}, o_{i,j})$, given that they have the admissibility property with regards to the bound they are applied to. As an example, we might in some game know from the material strength on the board in some state that we are guaranteed at least a draw, allowing us to initialize the pessimistic value to a draw. Similarly, we should be able to set the optimistic value to a draw if the opponent is equally up in material.

1	2	3	4	5
6	10	11	12	13
7	14	17	18	19
8	15	20	22	23
9	16	21	24	25

Figure 6: L-shaped cell ordering for 5×5 matrices.

Additionally, the order in which the pairs (a, b) will be visited in line 11 in Algorithm 1 can dramatically affect the amount of pruning. This problem can be decomposed into two parts. *Move ordering* in which the individual moves are ordered and *cell ordering* in which the joint moves are ordered based on the order of the individual moves. Formally, move ordering means endowing the sets A and B with total orders $<_A$ and $<_B$ and cell ordering is the construction of a total order for $A \times B$ based on $<_A$ and $<_B$. For instance, the lexicographical ordering is a possible cell ordering: (a_1, b_1) will be explored before (a_2, b_2) iff $a_1 <_A a_2$ or $a_1 = a_2$ and $b_1 < b_2$. We will discuss heuristic cell orderings in the next section.

5 Experimental Results

As a test case we implemented SMAB pruning for the game of Goofspiel. The following experimental results were obtained running OCaml 3.11.2, g++ 4.5.2, and the glpk 4.43 LP-solver under Ubuntu on a laptop with Intel T3400 CPU at 2.2 GHz.

5.1 Goofspiel

The game Goofspiel (Ross 1971; Shaei, Sturtevant, and Schaeffer 2009) uses cards in three suits. In the version we use, each player has all the cards of a single suit and the remaining suit is stacked on the table face up in a pre-defined order. On each turn both players simultaneously play a card from their hand and the higher card wins its player the top card from the table. If the played cards are of equal value the table card is discarded. When all cards have been played the winner is the player whose accumulated table cards sum up to a higher value. We used games with various number of cards per suit to monitor how the pruning efficiency develops with increasing game-tree sizes.

We use a cell ordering that strives to keep a balance between the number of rows filled and the number of columns filled. We call it *L-shaped* and it can be seen as the lexicographical ordering over tuples $(\min\{a, b\}, a, b)$. Its application to 5×5 matrix is described in Figure 6. In all of our preliminary experiments, the L-shaped ordering proved to lead to earlier and more pruning than the natural lexicographical orderings.

To save some calculations, it is possible to skip the LP computations for some cells and directly set the correspond-

Table 2: Solving GOOFSPIEL with backward induction.

size	nodes (= LP calls)	total time	LP time
4	109	0.008	0.004
5	1926	0.188	0.136
6	58173	5.588	4.200
7	2578710	247.159	184.616

Table 3: Solving GOOFSPIEL with a sequence form solver.

size	memory	time
4	8 MB	$<$1 s
5	43 MB	152 s
6	$>$ 2 GB	$>$ 177 s

ing α and β bounds to (minval - 1) and (maxval+1), respectively. On the one hand, if the computed bounds wouldn't have enabled much pruning, then using the default bounds instead allows to save some time. On the other hand, if too many bounds are loose, there will be superfluous computations in prunable subtrees.

To express this tradeoff, we introduce the *early bound skipping* heuristic. This heuristic is parameterized by an integer s and consists in skipping the LP-based computations of the α and β bounds as long as the matrix does not have at least s rows and s columns completely filled. For instance, if we use this heuristic together with the L-shaped ordering on a 5×5 matrix with parameter $s = 1$, no LP computation takes place for the bounds of the first 9 cells.

In our backward induction implementation that recursively solves subgames in depth-first-fashion, we used one LP call per non-terminal node expansion. Table 2 shows the number of non-terminal node expansions/LP calls, the total time spent running the algorithm, and the time spent specifically solving LPs.

Table 4 shows the same information for SMAB using L-shaped ordering and early bound skipping parameterized by s. This table has separate columns for the number of non-terminal node expansions and the number of calls to the LP solver as they are not equal in the case of SMAB.

Table 3 shows the memory and time needed to solve GOOFSPIEL using a sequence form solver based on based on (Koller, Megiddo, and von Stengel 1994). The algorithm needs a huge amount of memory to solve even a moderate size instance of GOOFSPIEL. The backward induction and the SMAB implementations, on the contrary, never needed more than 60 MB of memory. This difference is expected as the backward induction and SMAB are depth-first search algorithms solving hundreds of thousands of relatively small LPs while the sequence form algorithm solves a single large LP.

6 Conclusion and Future Work

We have shown that it is possible to extend Alpha-Beta pruning to include simultaneous move games and that our SMAB pruning procedure can reduce the node count and run-time when solving non-trivial games. In the reported experiments

Table 4: Solving GOOFSPIEL with SMAB.

size	nodes	LP calls	total time	LP time	s
4	55	265	0.020	0.016	0
4	59	171	0.012	0.012	1
4	70	147	0.012	0.012	2
5	516	2794	0.216	0.148	0
5	630	**1897**	**0.168**	0.128	1
5	1003	1919	0.184	0.152	2
6	13560	74700	5.900	4.568	0
6	18212	**55462**	**4.980**	3.852	1
6	30575	57335	5.536	4.192	2
7	757699	4074729	324.352	245.295	0
7	949521	2857133	259.716	197.700	1
7	1380564	2498366	241.735	182.463	2
7	1734798	**2452624**	**237.903**	177.411	3
7	1881065	2583307	253.476	188.276	4

we used a fixed move ordering and a fixed cell ordering. The results show a considerable drop in node expansions, even though not nearly as much as with Alpha-Beta in the sequential setting, but certainly enough to be very promising. Still, this threshold is not high and with increasing game size the run-time appears to be increasingly improving. The pruning criterion we propose is sound, but it only allows us to prune provably dominated actions. As a result, some actions which are not part of any equilibrium strategy may not get pruned by our method. Consider the following example:

Example 2. *The following game has a unique Nash equilibrium at* (A_2, B_2)*, but no action is dominated.*

	B_1	B_2	B_3
A_1	*6*	*1*	*0*
A_2	*3*	*3*	*3*
A_3	*0*	*1*	*6*

SMAB yields considerable savings in practice, but this example shows that there is room for even more pruning.

It will be interesting to see how SMAB pruning performs in other domains and it can also be applied to MCTS which has become the state-of-the-art algorithmic framework for computer GO and the GENERAL GAME PLAYING competition. A natural candidate is to extend the score bounded MCTS framework that we described earlier.

In our implementation we just used a naive move ordering scheme. However, simultaneous moves offer some interesting opportunities for improvements. As each individual action is considered more than once in a state, we get some information on them before their pairings are fully enumerated. The question is whether we can use this information to order the actions such that the efficiency of the pruning increases, like it does for sequential Alpha-Beta search.

Finally, it may be possible to establish the minimal number of node expansions when solving certain classes of stacked matrix games with depth-first search algorithms in general, or SMAB in particular.

Acknowledgments

We want to thank Martin Müller, Yannick Viossat, and Tim Furtak for frutifull discussions regarding SMAB, Marc Lanctot for letting us use his sequence form solver, and the anonymous reviewers for their constructive feedback. Financial support was provided by NSERC.

References

Ballard, B. W. 1983. The *-minimax search procedure for trees containing chance nodes. *Artificial Intelligence* 21(3):327–350.

Buro, M. 2003. Solving the Oshi-Zumo game. In van den Herik, H. J.; Iida, H.; and Heinz, E. A., eds., *Advances in Computer Games, Many Games, Many Challenges, 10th International Conference*, volume 263 of *IFIP*, 361–366. Graz, Austria: Kluwer.

Cazenave, T., and Saffidine, A. 2011. Score bounded Monte-Carlo tree search. In van den Herik, H.; Iida, H.; and Plaat, A., eds., *Computers and Games*, volume 6515 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 93–104.

Everett, H. 1957. Recursive games. *Contributions to the Theory of Games III* 39:47–78.

Furtak, T., and Buro, M. 2010. On the complexity of two-player attrition games played on graphs. In Youngblood, G. M., and Bulitko, V., eds., *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010*.

Gilpin, A., and Sandholm, T. 2007. Lossless abstraction of imperfect information games. *Journal of the ACM* 54(5).

Knuth, D. E., and Moore, R. W. 1975. An analysis of alpha-beta pruning. *Artificial Intelligence* 6(4):293–326.

Koller, D.; Megiddo, N.; and von Stengel, B. 1994. Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, 750–759.

Kovarsky, A., and Buro, M. 2005. Heuristic search applied to abstract combat games. In *Canadian Conference on AI*, 66–78.

Ross, S. M. 1971. Goofspiel: The game of pure strategy. *Journal of Applied Probability* 8(3):621–625.

Russell, S. J., and Norvig, P. 2010. *Artificial Intelligence — A Modern Approach (3rd international edition)*. Pearson Education.

Schiffel, S., and Thielscher, M. 2010. A multiagent semantics for the game description language. In Filipe, J.; Fred, A.; and Sharp, B., eds., *Agents and Artificial Intelligence*, volume 67 of *Communications in Computer and Information Science*. Springer Berlin Heidelberg. 44–55.

Shaei, M.; Sturtevant, N.; and Schaeffer, J. 2009. Comparing UCT versus CFR in simultaneous games. In *Proceedings of the IJCAI-09 Workshop on General Game Playing (GIGA'09)*, 75–82.

Sturtevant, N. R., and Korf, R. E. 2000. On pruning techniques for multi-player games. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, AAAI/IAAI 2000, 201–207.

Sturtevant, N. R. 2005. Leaf-value tables for pruning non-zero-sum games. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, 317–323. Edinburgh, Scotland, UK: Professional Book Center.

Winands, M. H.; Björnsson, Y.; and Saito, J.-T. 2008. Monte-Carlo tree search solver. In *Proceedings of the 6th international conference on Computers and Games*, CG '08, 25–36. Berlin, Heidelberg: Springer-Verlag.