# AN ALGORITHMIC SOLUTION OF N-PERSON GAMES

Carol A. Luckhardt and Keki B. Irani

Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan 48104

## ABSTRACT

Two-person, perfect information, constant sum games have been studied in Artificial Intelligence. This paper opens up the issue of playing n-person games and proposes a procedure for constant sum or non-constant sum games. It is proved that a procedure, $\max^n$, locates an equilibrium point given the entire game tree. The minimax procedure for 2-person games using look ahead finds a saddle point of approximations, while $\max^n$ finds an equilibrium point of the values of the evaluation function for n-person games using look ahead. $Max^n$ is further analyzed with respect to some pruning schemes.

## I *INTRODUCTION*

Game playing is one of the first areas studied in Artificial Intelligence (AI) [Ric83]. Most of the work has been done with games that are 2-person, finite, constant sum (and therefore non-cooperative), perfect information and without a random process involved. For example, chess and checkers involve two people, have a finite number of strategies available to each player, pay the same total amount at the end of the game, each player knows the other player's moves, and there is no chance involved. The most famous game programs are the chess players such as the Cray-Blitz, Chaos and Belle [Nel84]. This paper addresses n-person games, that is, games with more than two players, and describes a method of computer play for non-cooperative, non-constant sum games, and for cooperative games given a coalition structure. The approach has been to bring game theoretic results into the more pragmatic AI domain.

## II *BACKGROUND*

Trees are often used as models of decision making in AI and in game theory. From the rules or definition of a game, the game tree representation can be specified for an n-person game by a tree where [Jon80]:

(1) the root node represents the initial state of the game,
(2) a node is a state of the game with the player whose move it is attached to it,
(3) transitions represent possible moves a player can make to the next possible states,
(4) outcomes are the payoff assignments associated with each terminal node, which are n-tuples where the $i^{th}$ entry is paid to player i.

Because most games of interest have combinatorially explosive game trees, AI programs tend to analyze partial game trees in order to determine a best move. An *evaluation function* is a function which estimates what resulting value the game should have when given a terminal node of a partial game tree. Then by the *look ahead* procedure, values are backed up from the terminal nodes to each node of the tree according to the *minimax* searching method [Ric83]:

(1) at the program's move, the node gets the maximum value of its children,
(2) at the opponent's move, the node gets the minimum value of its children.

The value that is backed up to the root node is the value of the game, and the move taken should be to a node that has that value as its backed up value. If the whole tree is available to be analyzed, there is a theorem from game theory called the *minimax theorem* [LuR57] that applies. It is for 2-person zero sum games. *Zero sum* means that the payoff values for each player add up to zero for any payoff vector. The theorem says there is a strategy that exists for each player that will guarantee that one gets at most v while the other loses at most v and the value of the game is v. This set of strategies, one for each player, is called a *saddle point.*

For example, in the game of 2-2 Nim, initially there are 2 piles of 2 tokens. Players A and B alternate turns. Each player selects a pile and removes any number of tokens from that pile, taking at least one. The loser is the one who takes the last token.

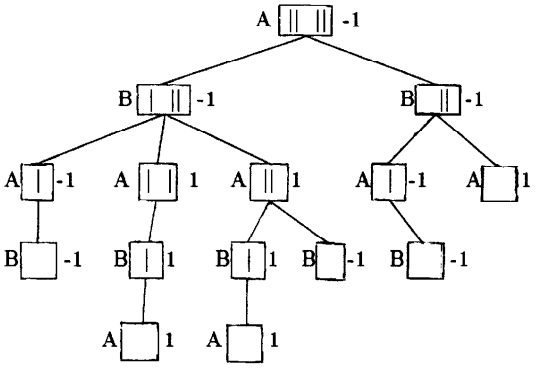


Figure 1. 2-2 Nim

The terminal node value of 1 corresponds to the vector (1,0) and -1 corresponds to (0,1). Since this is a 2-person zero sum game, the outcomes can be represented by one number. The value of 2-2 Nim is -1 which means that no matter what A does, B can always make a move that will lead to a win for B.

A technique from AI called *alpha-beta pruning* [Ric83] reduces the number of nodes that have to be visited when calculating the minimax values. For example, in the above game tree ordering, when doing a depth first search and backing up to B [ | ||], the left most child needs to be evaluated to get a -1 and then it is not necessary to look any further since this is the best that B can do. If a game tree has depth d and branching factor b, then in the best case of this pruning procedure, $2b^{d/2}$ nodes are evaluated rather than the complete $b^d$ nodes [Win77].

## III *N-PERSON GAMES*

Considering games with more than two players, one value will no longer suffice in representing the outcome. A vector is required for both constant and non-constant sum games. A *constant sum* game is one where the sum of the entries in an outcome vector is the same value for any terminal node. It no longer makes sense to evaluate the game based on any one player's payoff values.

Game theory solutions to non-cooperative games are usually a set of strategies for each player that are in some sense optimal, where the player can expect the best outcome given the constraints of the game and assuming the other players are attempting to maximize their own payoffs. A solution for an n-person, perfect information game is a vector which consists of a strategy for each player, $(s_1, \ldots, s_n)$. A strategy defines for the player what move to make for any possible game state for the player. Call the set of possible strategies for player i, $P_i$, and the payoff to player i, $U_i$. $U_i$ is a real valued function on a set of strategies, one for each player. The set $\{P_1, \ldots, P_n; U_1, \ldots, U_n\}$ is called the *normal form* of a game [Jon80]. An *equilibrium point* for $\{P_1, \cdots, P_n; U_1, \cdots, U_n\}$ is a strategy n-tuple $(s_1, \cdots, s_n)$, such that for all i=1,...,n and $s_i, s_i' \in P_i$,

$$U_i(s_1, \ldots, s_i', \ldots, s_n) \leq U_i(s_1, \ldots, s_i, \ldots, s_n).$$

The $s_i$'s are called *equilibrium strategies*. For example, in the game represented by,

| | $\beta_1$ | $\beta_2$ |
|---|---|---|
| $\alpha_1$ | (-4,-4) | (1,-9) |
| $\alpha_2$ | (-9,1) | (-1,-1) |

Figure 2. 2X2 game

where player A's strategies are the $\alpha_i's$ and B's are the $\beta_i's$, and (a,b) means pay a to the first player and b to the second player, $(\alpha_1, \beta_1)$ which corresponds to (-4,-4) is an equilibrium point. The equilibrium point has the property that no player can improve his or her expected payoff by changing his or her own choice of strategy if the other strategies are held fixed. A saddle point is an equilibrium point, while an equilibrium point may not be a saddle point.

These non-cooperative games with perfect information are always solvable in this sense according to the following theorem.

Theorem 1:

A finite n-person non-cooperative game which has perfect information possesses an equilibrium point in pure strategies (proof in [Jon80], page 63).

A *pure* strategy is a single $\alpha_i$ or $\beta_i$, as we have seen so far. The theorem just states the existence of an equilibrium point, not how to find one.

## IV *MAX^N*

If we have rational players who are trying to maximize their own payoffs, the backed up values should be the maximum for each player at each player's turn. We call this procedure *Max^n*. The max^n procedure, maxn(node), is recursively defined as follows:

(1) For a terminal node,
maxn(node) = payoff vector for node

(2) Given node is a move for player i, and
$(v_{1j}, \ldots, v_{nj})$ is
maxn($j^{th}$ child of node), then
maxn(node) = $(v_1^*, \ldots, v_n^*)$,
which is the vector where $v_i^* = \max_j v_{ij}$.

Calling the procedure with the root node finds the max^n value for the game and determines a strategy for each player, including a move for the first player. This procedure can be used with a look ahead where a terminal node in the definition above becomes a terminal node in the look ahead. For example, given the payoff vectors on the bottom row, by the procedure, A should take the move represented here by the right child:

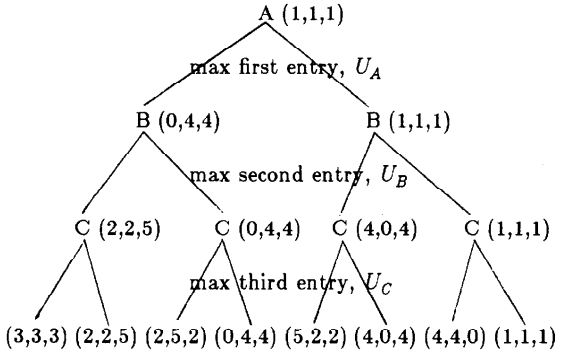


Figure 3. max^n example

Note that this procedure does not require that there be an order in the moves of the players going down the tree. For example, B could follow C.

The next theorem shows that max^n finds an equilibrium point. There may be more than one equilibrium point. When a tie occurs in the back up, each possible choice will lead to an equilibrium point, so it does not matter which move is selected.

Theorem 2:

Given an n-person, non-cooperative, perfect information game $\{P_1,...,P_n; U_1,...U_n\}$, in tree form, max^n finds an equilibrium point for the game.

Proof:

Backing up values in the tree by applying the max$^n$ procedure, with some tie breaker, determines a strategy for each player which gives a strategy set S=($s_1$,...,$s_n$), $s_i \in P_i$, i=1,...,n. So, at each node for each player i, the strategy $s_i$ gives the arc or move choice which maximizes the backed up value of $U_i$ of the children nodes. In order to have an equilibrium point, we need to show that for all i,

$U_i(s_1,...,s_i,...,s_n) \geq U_i(s_1,...,s_i',...,s_n)$, for all $s_i' \in P_i$.

Suppose that there is some $s_j' \in P_j$, $s_j' \neq s_j$ where this is not true. That is,

$U_j(s_1,...,s_j,...,s_n) < U_j(s_1,...,s_j',...,s_n)$.

The strategy set S'= $\{s_1, \ldots, s_j', \ldots, s_n\}$ differs from S=$\{s_1, \ldots, s_j, \ldots, s_n\}$ in the tree only at the nodes where it is j's turn. As we work from the terminal nodes up the tree on the path defined by max$^n$, $s_j'$ must change this path or the payoff would be the same. Let us consider the place where $s_j$ and $s_j'$ first differ:

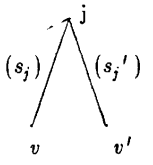


Figure 4. where the strategies differ

where $v=(v_1, \ldots, v_n)$ and $v'=(v_1', \ldots, v_n')$ are the max$^n$ backed up values which are the payoffs for the strategy sets S and S', respectively, and $v_i=U_i(s_1, \ldots, s_i, \ldots, s_n)$, $v_i'=U_i(s_1, \ldots, s_i', \ldots, s_n)$. From our assumption $v_j < v_j'$ but by the max$^n$ procedure $v_j \geq v_j'$. This contradiction proves the theorem. □

An equilibrium point exists according to theorem 1, and it is the best a player can do if the opponents are rational, which means taking the maximum of the utilities available to them. This procedure seems a likely candidate for playing n-person, non-cooperative, perfect information games in the AI domain, that is, games to be played intelligently by a computer. Just as the minimax procedure with an evaluation function approximates a saddle point in two person, perfect information games, if we use max$^n$ with a good evaluation function, we can approximate an equilibrium point. Actually we would be finding an equilibrium point of the approximations given by the heuristic function. It is also possible to check each point and analyze it to see if it might be an equilibrium point. *Max$^n$* gives a quick result on which to base a move choice.

An estimated payoff calculation for a node does not need to be for the whole vector. The value needed immediately is the estimated payoff for the entry of the player of the parent node in order to make a comparison to decide which value to back up. We will consider types of possible pruning related to this.

## V *SHALLOW PRUNING*

Since in searching for the max$^n$ value a maximum is always sought after, pruning of subtrees as in alpha-beta is not possible. However, some pruning of individual payoff entries within the vector is possible if the entries are calculated separately. A simple pruning would be to calculate the entire vector only for the best child of the terminal nodes. Only one entry from the other payoff vectors is needed. First, evaluate the payoff entry for the parent node in each of the children and find the maximum entry. Then back up the entire vector of that child. If a game tree has a constant branching factor b and we look ahead m levels, which would usually be a multiple of n, then the number of evaluations is $nb^m$, without any pruning. With this *simple shallow pruning,* rather than evaluating all $nb^m$ numbers, only one vector entry value for each $b^m$ terminal plus the rest of the vector for the best child of each of the $b^{m-1}$ parents is calculated. Thus, the number of evaluations is $b^m+(n-1)b^{m-1} = b^{m-1}(b+n-1)$. The percentage of entries evaluated is:

$$\frac{b^m+nb^{m-1}-b^{m-1}}{nb^m} = \frac{1}{n}+\frac{1}{b}-\frac{1}{nb}.$$

Note that this does not depend on the number of levels being searched.

A further improvement on this is to calculate a value only when it is needed for the next comparison. Instead of only for terminal nodes as in the simple shallow pruning, do this for all levels of nodes. Each time a child's values are backed up, the next value to the left in the vector of payoffs needs to be calculated. That is, the payoff for the player a level above needs to be calculated from the terminal node from which the backed up value came. Call this *shallow pruning* for n-person games. The number of evaluations out of $nb^m$ done with this type of pruning is:

$$b^m+b^{m-1}+\cdots+b^{m-(n-1)}$$

$$= b^m+\cdots+b+1 - (b^{m-n}+b^{m-(n+1)}+\cdots+b+1)$$

$$= \frac{b^{m+1}-1}{b-1} - \frac{b^{m-n+1}-1}{b-1}$$

$$= \frac{b^{m+1}-b^{m-n+1}}{b-1}$$

The following procedure returns the entry payoff of the max$^n$ vector and determines the strategy for the player of node as a side effect. The max$^n$ algorithm with shallow pruning is:

```
pmaxn(node);  /* returns maxn value */
  BEGIN
  IF node is terminal in the look ahead
  THEN evaluate and return the parent's payoff
  ELSE
      BEGIN
      FOR each child of node
      DO BEGIN
          v := pmaxn(child)
          IF v is the best value of the children
          THEN back up the value and child pointer
          END
      calculate the value for the grandparent of
        the best child and back it up also
      RETURN v
      END
  END
```

The algorithm is illustrated in the example in the next section.

The number of comparisons done of payoff values with any of these searches is the same. At the lowest level where the terminal nodes are, for each of the $b^{m-1}$ sets of children, there are b-1 comparisons made, and b-1 for each of the $b^{m-2}$ groups of b nodes above, and so on, to the final b-1 comparisons at the root node. So, the number of total comparisons is:

$$b^{m-1}(b-1)+b^{m-2}(b-1)+\cdots+b(b-1)+(b-1)$$

$$= \frac{b^m-1}{b-1}\cdot(b-1)$$

$$= b^m-1$$

## VI *EXAMPLE*

As an example of shallow pruning, see figure 5 for the first three moves of the 2-2-2 Nim game for three players.

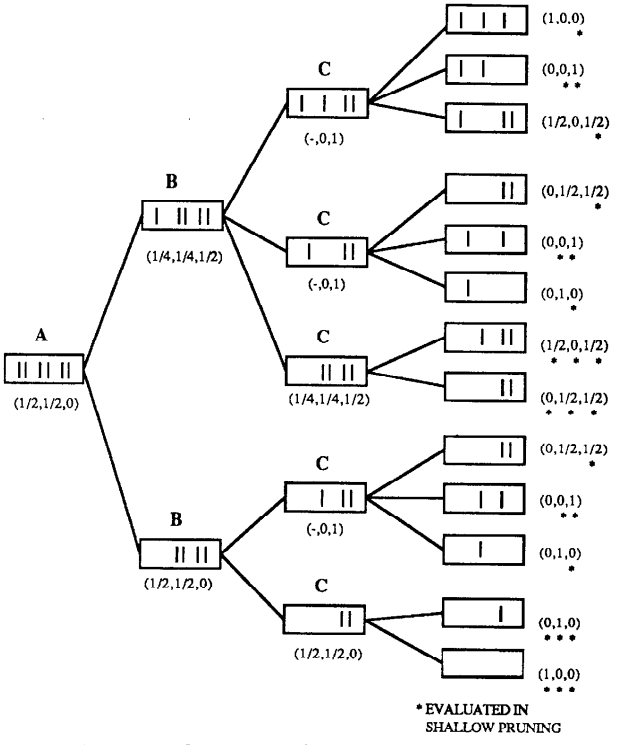


**Figure 5. three person 2-2-2 Nim with three levels of look ahead**

The game is played just like the other Nim games. Players alternate turns taking one or more pieces from any one group. The goal can be varied to give a different evaluation and strategy for playing. The goal in this case is to have the player before you take the last piece. The player who achieves the goal gets 1 unit of reward and the other two get nothing. The evaluation function used for the look ahead estimate is the percentage of the possible number of moves left in the game which leads to a win for the player. To calculate that, first find the minimum number of moves left in the game which is equal to the number of groups, say a for example. Then find the maximum number of moves left, which is equal to the total number of pieces left, b for example. The possible number of moves in the game ranges from a to b, or the possibilities are a, a+1, a+2, ... , b-1, b. The estimated payoff in the look ahead player for A is the number of these that are divisible by 3 ( $= 0 \bmod 3$), divided by $|\{a,a+1, \ldots ,b-1,b\}| = b-a+1$. The estimated payoff for B is the percentage of the numbers that are equal to 1 mod 3, and for C it is equal to 2 mod 3.

For example, with one group of one piece and one group of two pieces we have a=2 and b=3. The estimated payoffs for A, B, and C are 1/2, 0/2=0, 1/2, respectively. In the example given, an exhaustive search would require 39 evaluations while the shallow pruning requires 24 evaluations, or 62% of an exhaustive search. The back up procedure suggests that A should take the lower child in the representation for its first move. Ties are handled by backing up the average of the payoffs for each player, which is a possible variation with which to play. Note that when this is done, more evaluations may be needed than the stated formula suggests.

## VII *DEEP PRUNING*

The pruning described here could be correlated to a deep cutoff which was made distinct from a shallow cutoff by Pearl [Pea84]. A deep cutoff uses information from great grandparent nodes. When a value is backed up, the entry for the player of the grandparent node must also be sent for the comparison at the next level up. A *deep pruning* procedure for n-person games is:

```
(1) evaluate the far left, lowest level children
        for the last player's payoff,
    find the best of the components,
    evaluate the best vector,
        (v_1, . . . , v_n), and
    back it up one level
(2) IF at the root node
    THEN return the vector
    ELSE BEGIN
        back the vector up one level to player i
        FOR each unvisited terminal node below
        DO BEGIN
            IF v_i < the payoff to player i
                    at the terminal node
            THEN back up the best vector
                    by shallow pruning
            END
        REPEAT (2) with the backed up node
        END
```

Applying deep pruning to the example used for shallow pruning requires seven more evaluations than the shallow pruning. The game tree in figure 3 requires 16 evaluations in simple shallow, 14 in shallow and 19 in deep pruning. Figure 6 is an example which benefits from deep pruning. The second set of payoffs shows which entries are evaluated. There are 10 evaluations with deep pruning verses 14 with shallow.

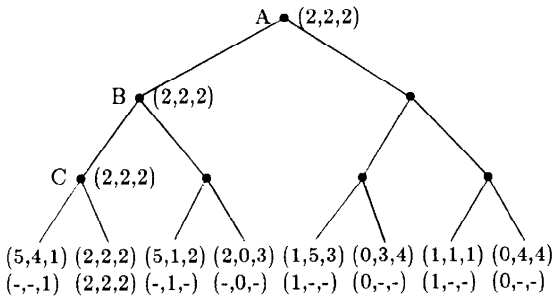


Figure 6. deep pruning

The best case, shown above, would evaluate $b^m+n-1$ values in the general n-person game tree with constant branching factor b, m levels of look ahead, and n players. This is better than the case for shallow pruning. Deep pruning would be very useful if some predictable order of terminal nodes were available.

In the worst case, at each check going down the tree, the comparison would call for a different vector value to be backed up. In that case, below each of the $b^{m-1}$ nodes in the level next to the bottom, the number of evaluations required is:

n values for the vector +
b-1 values to find the best child +
b-1 values of the deep pruning check which would fail on the last node checked in the worst case.

Adding this up for the $b^{m-1}$ nodes, and subtracting (b-1) since the first set of children is only evaluated to find the best child and not a deep pruning check, we get:

$$b^{m-1}[n+2(b-1)]-(b-1) =$$
$$2b^m+(n-2)b^{m-1}-b+1 =$$
$$(b^m+nb^{m-1}-b^{m-1})+(b^m-b^{m-1}-b+1).$$

This last expression is the number of evaluations in simple shallow pruning plus $(b^m-b^{m-1}-b+1) = (b^{m-1}-1)(b-1)$.

## VIII *COOPERATIVE GAMES*

A *cooperative* game is one in which communication and coalition formation is allowed between players. A *coalition* is a subset of the n players such that a binding agreement exists between the players. The coalition can be treated as one player with a strategy which is collectively determined. When it is a player's turn who is in the coalition, it is the coalition's move. A *coalition structure* on an n-person game $\{P_1, \ldots, P_n; U_1, \ldots, U_n\}$ is a partition of {1,...,n}. Call the partition $S=\{S_1, \ldots, S_m\}$, where

$$S_i = \{q_{i1}, \ldots, q_{iz_i}\},\ q_{ij} \in \{1,\ldots,n\},$$
$$S_i \bigcap S_j = \phi \text{ for all } i \neq j, \text{ and}$$
$$S_1 \bigcup \cdots \bigcup S_m = \{1,\ldots,n\}$$

We can now use max$^n$ for cooperative games by the following theorem.

Theorem 3:

For any coalition structure $\{S_1, \ldots, S_n\}$, $S_i = \{q_1, \ldots, q_{z_i}\}$, on a cooperative n-person game $\{P_1, \ldots, P_n; U_1, \ldots, U_n\}$, max$^n$ finds an equilibrium point for the m-person non-cooperative game $\{R_1, \ldots, R_m; W_1, \ldots, W_m\}$ where $R_i = P_{q_1} \times \cdots \times P_{q_{z_i}}$, and $W_i(r_1, \ldots, r_m) = \sum_{j=1}^{z_i} U_{q_j}(s_1, \ldots, s_n)$, $r_k \in R_k, s_k \in P_k$.

Proof:

Apply theorem 2 to the non-cooperative game $\{R_1, \ldots, R_m; W_1, \ldots, W_m\}$. In the tree form, it is $R_i$'s turn whenever it is a player's turn who is in the coalition $R_i$.

Assuming a coalition structure has been determined and will remain constant for a cooperative game, max$^n$ can be applied to the resulting non-cooperative game with a meaningful result. *Max$^n$* can be used in determining a move for a computer in n-person games under these conditions.

## IX *CONCLUSIONS*

As an answer to how should a computer play n-person, non-cooperative games, max$^n$ with pruning is a satisfactory approach given a good evaluation function. In the best case situation, deep pruning does the least number of evaluations, but in the worst case for deep pruning, it does worse than even the simple shallow pruning. Shallow pruning does fewer evaluations than simple shallow pruning, however, more traveling by pointers in the tree is required. For cooperative games with a given coalition structure, max$^n$ will find an equilibrium point as a possible solution of the game and determine a strategy for a coalition. Using this approach, we are looking at the question of what are the best coalitions to be formed. The max$^n$ algorithm might also be applied to imperfect information games or games with chance involved.

## REFERENCES

[Jon80] Jones, A.J. *Game Theory: Mathematical Models of Conflict.* West Sussex, England: Ellis Horwood, 1980.

[LuR57] Luce, R., and Raiffa, H. *Games and Decisions.* New York: John Wiley & Sons, 1957.

[Nel84] Nelson, Harry. "How we won the Computer Chess World's Championship" In *Lawrence Livermore National Laboratories Tentacle,* (excerpt from DAS Computer Science Colloquium). LLNL, Livermore, CA, January 1984.

[Pea84] Pearl, Judea. *Heuristics.* Massachusetts: Addison-Wesley, 1984.

[Ric83] Rich, Elaine. *Artificial Intelligence.* USA: McGraw-Hill, 1983.

[Win77] Winston, Patrick H. *Artificial Intelligence.* Reading, Mass.: Addison-Wesley, 1977.