CS324: Adversarial Search and Game Theory. **Spring 2015**

Lecture 18: April 16, 2015

Lecturer: K.R. Chowdhary *: Professor of CS (VF)*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

18.1 Introduction

The Game theory is the formal study of *conflict* and *cooperation*. Game theory's concepts apply whenever the actions of several agents are interdependent. These agents may be individuals, groups, firms, or any combination of these. The concepts of game theory provide a language to formulate, structure, analyze, and understand strategic scenarios. Cooperative game theory concentrate on what agreements the agents are likely to reach, while noncooperative game theory concentrate on what strategies are likely to be adopted by the agents. Both of these traditions have adopted models of bargaining in which each agent's risk posture is conveyed by comparing his preferences for risky and riskless alternatives.

Another branch of game theory, called, *combinatorial Game theory* (CGT) studies strategies and mathematics of two-palyer games of *perfect knowledge* such as *chess* or *go*. But generally concentrates on simpler games like *nim* or solving *end-games* or their special cases. An important difference between this subject and *classical game theory* (a branch of economics, after von Neumann and John Nash) is that game players are assumed to move in sequence rather than simultaneously, so their is no point in randomization or information hiding strategies.

The CGT does not study with importance in computer knowledge, called games of chance, like poker (card games). The combinatorial games include, games like chess, checkers, Go, Arimaa, Hex, and Connect. They also include one player combinatorial puzzles, and even no-player automata.

In CGT, moves are represented as game-tree, and the trees can be searched while making moves in the game. A game-trees is a special kind of semantic tree where nodes represent board configurations and branches indicate how one board configuration can be transformed into another configuration by a single move. The decisions regarding the next move are made by two *adversaries*, who play the game, hence the name *adversarial search*.

18.2 Classification of Games

Games are classified into several categories based on certain significant features, the most obvious of which is the number of players involved. A game can thus be designated as one-person, two-person, or n-person ($n > 2$). A player, i.e., a participant in a game need not to be a single person. If each member of a group has same feelings about how the game should progress, and end, the members may be called as a single player. A player may also be a corporation, or a basketball team. or even a country.

In games of perfect information, such as chess, each player knows every thing about the game at all times. Poker, on the other hand is a game of imperfect information because players do not know the the cards the others are having.

The extent to which the goals of the players are opposed or coincidence, is another basis of classifying the games. The zero-sum or to be precise constant-sum games are completely competitive. The Poker, for example, is zero-sum because because the combined wealth of the players remains constant; if one player wins the other must lose. In nonzero-sum games all players can be winners or losers. In a dispute between management v/s labor, if some agreement is not reached, both will be losers. For example, if labor strike cannot be avoided.

In cooperative games, the bidders may communicate and make agreement as binding to both the players, and in noncooperative they may not. The examples of players in cooperative games are: car salesman v/s customer, management v/s employees, where as, bidders for government tender are cooperative bidders.

18.3 Game Playing Strategy

Let there be a facility by which every unique board configuration can be converted into a unique single, overall quality value in the form of an integer number. Further, consider that positive number indicate the move in favour of one player, and negative - for the favour of his/her opponent. The degree of favouredness is absolute value of that number. Let us call ourself as *maximizer* player, and the opponent as *minimizer*. Also, let us imagine that, there is a number (call it static *score*, accumulated so far), the maximizer will make a move such that the score increases to maximum, while the opponent (adversary) makes a move so that by addition of quality number, the score number minimizes.

The game playing is favorite for AI search, because:

- It is structured task, often a symbol of "intelligence",
- clear definition of success and failures,
- does not require large amounts of knowledge (at first glance),
- focus on games of perfect information, and
- multi-player, and chance game.

The difference between the games and search problems is that the games playing are highly unpredictable, as one does not know what move the opponent is going to make. And, only based on the opponent's move the other player has to make a [1]. The other difference is that the time limits for a move are in realistic times, hence one is unlikely to find goal in that time, therefore approximations are necessary for both the players. The formal system behind these games is called *Game-Theory.*

18.4 Two-person zero-sum Games

The term "zero-sum" (or equivalently "constant sum") means that players have diametrically opposite opposed interests. The term comes from parlor games like poker where there is fixed amount of money around the table. If one player wins some money, other have to lose an equivalent amount. Two nations trading make a non-zero-sum game since both are simultaneously in gain. An equilibrium point is a stable outcome of game associated with pair of strategies.

[1] A player can learn from the opponent's past moves as how and what strategy the opponent followed. But, this requires the learning ability. Hence, game's moves shall not be based current state, but past also. This has not been considered for the present discussion.

Consider the figure 18.1, which is played between you and your opposite. The numerical values in the cells indicate the figures in dollars your opponent will pay to you. To start the game, assume that you select row B. Next, your opponent will select a column, say II. Therefore, your opponent will pay you \$4. Next, say, you select row C, your opponent, suppose selects column III. In this case you pay to your opponent one dollar, because the value at the junction is -1. The two-player zero-sum game is a complete information game.

A strategy in a game theory is a complete plan of action that describes what a player will do under all possible circumstances. There are poor strategies and good strategies.

		YOUR OPPONENT		
		I	*II*	*III*
	A	5	-2	1
YOU	*B*	6	4	2
	C	0	8	-1

Figure 18.1: Two-person zero-sum game.

Other examples of two-person zero-sum games of perfect information are *chess*, *checker*, and Japanese *go*. Such games are strictly determined, i.e., rational players making use of all available information can deduce a strategy that is clearly optimal, hence the outcome of such games are deterministic. In chess, for example, one of these three possibilities exists: 1) white has an winning strategy, or 2) black has a winning strategy, or 3) black and white each have a strategy that is winning or draw.

18.4.0.1 Two-person nonzero-sum Games

Most games that arise in practice are nonzero-sum games, where players have both common as well opposed interests. For example, buyer vs. seller is a nonzero-sum game. The buyer wants a lower price and seller a higher price, but both want that deal be executed. Similar is the case with game of hostile countries, who may disagree on many issues but try to avoid the war. Many obvious properties of zeros-sum games are not available in nonzero-sum games. One of the difference is effect of communication on game, which does not help the opponent in zero-sum game. The game of labour-union vs factory-owner is a nonzero-sum game. If the management is properly informed about the demands of labor union, probably, the strike may get withdrawn, benefiting both the laborer and factory owner. The games where players communicate, make binding agreement between two parties, are called *cooperative games*, and where players are not allowed to communicate are *non-cooperative games*. In a nonzero-sum game, unlike in figure 18.1, each table entry consists a pair of numbers. This is because, the combined wealth of the players is not constant. Since it is not possible to deduce one player's payoff from the payoff of the other player, both players payoff must be specified. The prisoner's dilemma is best example of nonzero-two player non-cooperative.

18.5 The Prisoner's Dilemma

Two criminals A and B are arrested and interrogated separately. Each suspect can either confess (*defect* the other prisoner) with a hope of a lighter sentence or refuse to talk (*cooperate* the other prisoner). The police does not have sufficient information to convict the suspects, unless at least one of them confesses. If they cooperate, then both will be convicted to minor offense and sentenced to a month in jail in the absence of supportive evidence. If both defect, then both will be sentenced to jail for six months. If A confesses and B does not, then the A will be set free but the B will be sentenced to one year in jail. The police explains these outcomes to both suspects and tells each one that the other suspect knows the deal as well. Each suspect must choose his action without knowing what the other will do. This is shown in a table in figure 18.2.

A close look at the outcomes of different choices available to the suspects reveals that regardless of what one suspect chooses, the other suspect is better off by choosing to defect, i.e., confesses. Hence, both suspects choose to defect and stay in jail for six months, opting for a clearly less desirable outcome than only a month in jail, which would be the case if both chose to cooperate each other by not confessing.

Suspect A \ Suspect B	Confess	Do not confess
Confess	(6 mon., 6mon.)	(0 mon., 1 yr.)
Do not confess	(1 yr., 0 mon.)	(1 mon., 1 mon.)

Figure 18.2: Prisoner's Dilemma.

We assume that there is no honor among the criminals, and each ones sole concern is to save himself. This game is called *prisoner's dilemma*. Since a suspect must make his own decision without knowing that of other, he must consider each of his partner's alternatives and anticipate the effect of those on him.

Supposes the A confesses, B must go to jail for one year or for six months. Alternatively, if A does not confess, then either B is set free or given one month's jail. In either case, for B it is better to confess !

However, if both cooperate to each other, both will not confess, and both go for miner term in jail of 1 month. If both confess, they go jail for 6 months each.

Further, we conclude that, both may be cooperative or uncooperative. In cooperative case each does better, than they and uncooperative. Also, for any fixed strategy of the opponent, other player does better by playing uncooperatively.

18.6 Games of Imperfect Information

The simplest two-person zero-sum games of imperfect information have saddle points. Such games have predetermined outcomes provided it is rational play. The predetermined outcome is called value of the game. Consider the table given in figure 18.3.

The two campaigning political parties, A and B must each decide how to handle a disputed issue in a village. They can support it, oppose it, or evade it. Each party must make a decision without knowing what its rivals will do. Each pair of decision (x, y), where x, y either of three decisions by party A and B, determine the percentage of votes that each party receives in this village. Each party wants to maximize its own percentage of the vote. The numbers in the matrix in figure 18.3 indicate the percentage share of votes for party A, the balance (100 minus this value) is that of party B. Suppose party A oppose the issue and B supports it, then A gets 80% and B gets 20%.

Party A \ Party B	support	oppose	evade
support	60	20	90
oppose	80	25	75
evade	35	30	40

Figure 18.3: Imperfect Information game.

The A's decision seems difficult at first, because it depends upon B's strategy. A does best to support if B evades; A does best to oppose if B supports; and A succeeds to evade if B opposes it. We note that party A must consider the B's strategy before deciding its own strategy. What ever A decides, B gets the largest share by opposing it. Interestingly, once this is realized by A, he prefers to settle for 30% ! This 30%:70% votes for $A : B$ is called game's *saddle point*.

A better way of finding the saddle point is to determine the *maximin* and and *minimax* values. Using this

method, first A determines the minimum percentage of votes it can obtain for each of its strategies, then selects the maximum of these three. The minimum percentage A will get if it supports, oppose, and evade are: 20, 25, 30, respectively. The largest of these, i.e. 30, is maximin (maximum of the minimums) value. Similarly, each strategy the B chooses, it determines the maximum of percentage of votes A can win, i.e. minimum B can win (similar to what we did for A). In this case, if B supports, oppose, and evade, the maximum A gets is: 80, 30, 90, respectively. Now, the B obtains its highest percentage by minimizing A's maximum percentage of vote. Here, smallest of A's maximums is 30. This 30 is B's minimax value. Because both the maximin and minimax are 30, hence 30 is saddle point.This selection process process is shown in figure 18.4.

18.7 Representation of two-player games

Chess has certain attractive features that many, more complex tasks do not have. The available options (moves) and goal (check-mate) are sharply defined. The discrete model of chess is called a game-tree, and it is the general mathematical model on which the theory of two-player zero-sum games of perfect information is based. It is perfect information because all legal moves are known to both players at all times, and it is zero sum because one player's loss equals the other's gain.

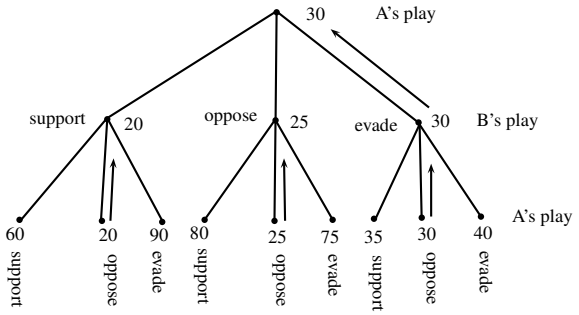


Figure 18.4: Selection of maximin and minimax.

At the top of the chess tree is a single root node, which represents the initial setup of the board. For each legal opening move, there is an arc leading to another node, corresponding to the board configuration after that move has been made. There is one arc leaving the root for each legal opening, and the nodes that they lead to define the setups possible after one move has been made. More generally, a game-tree is a recursively defined structure that consists of a root node representing the current state and a finite set of arcs representing legal moves. The arcs point to the potential next states, each of which, in turn, is a smaller game-tree. The number of arcs leaving a node is referred to as its branching factor, and the distance of a node from the root is its depth. If b and d are the average branching factor and depth of a tree, respectively, the tree contains at least b^d nodes. When the current state of the game is a leaf, the game terminates.

Each leaf has a value associated with it, corresponding to the payoff of that particular outcome. Technically, a game can have any payoff (say a dollar value associated with each outcome), but for most standard parlor games, the values are restricted to *win* and *loss* (and sometimes *draw*).

18.8 Minimax Search

Consider the figure 18.5, where the two players (*maximizer* and *minimizer*) play alternately. The game has static values at the lowest level, i.e., at the bottom of the tree. The game tree is shown with three levels 0 (for root), 1, and 2. The maximizer chooses move at level 0, minimizer at level 1, and at next level, i.e., 2, the static values are available. The maximizer might hope to get to the situation yielding score 9, he knows that minimizer might choose a move deflecting the play towards the score 2. In general, the maximizer (here

at level 0) must take the choices available to the minimizer at next level, into cognizance. And, similarly, the minimizer must also take into cognizance the choices available to maximizer at the next level down.

Eventually, the limits of exploration are reached and the static evaluator provides the direct basis for selecting among the alternatives. The minimizer may choose the values 3 and 2, at the level just up from the static evaluations(fig. 18.5(b)). Knowing these scores the maximizer at level one up makes the best choice between 2 and 3, at level zero(see fig. 18.5(c)).

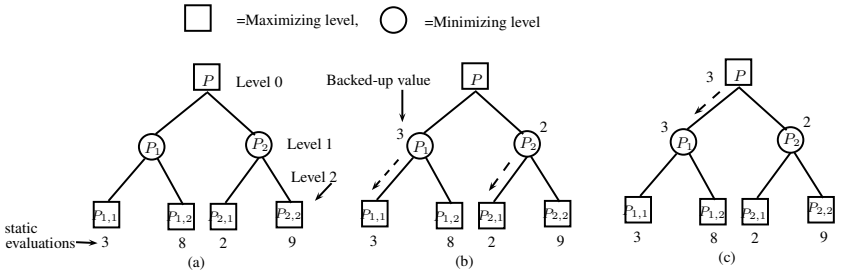


Figure 18.5: MINIMAX Search.

The procedure by which the scoring information passes up the game tree is called MINIMAX algorithm, because the score at any level i is either minimum or the maximum of what is available at the level $i+1$. The recursive form of the MINIMAX algorithm is shown as Algorithm 1. The idea of *minimaxing* is to translate the board into a *static number*. However, the process may be expensive due to generation of large number of paths.

Algorithm 1 MINIMAX-Algorithm

```
1: if limit of search has reached then
2:    compute the static value of current position relative to the appropriate player;
3:    Report the result;
4: else
5:    if level is minimizing level then
6:       call MINIMAX-Algorithm on the children of current position;
7:       Report minimum of the result;
8:    else
9:       (level is maximizing level.)
10:      call MINIMAX-Algorithm on the children of current position;
11:      Report maximum of the result;
12:   end if
13: end if
```

The *minimax* algorithm has the following properties:

1. The algorithm is *complete* if tree is finite.
2. The worst-case Time complexity is $O(b^d)$, where b is branching factor, and d is total distance.
3. The worst-case Space complexity is $O(d)$ equal to depth-first exploration.

18.9 Tic-tac-toe game analysis

The tic-tac-toe is a board game, with $3 \times 3 = 9$ positions, and played between two players through alternate moves. The game has total $9! = 362880$ configurations, and many of them collapses into single configurations due to symmetry, hence the actual number of configurations are far less. Initial configuration is empty board.

The one player, called as MAX makes the first move with X, and the other player, called MIN player, makes a O move. There are total three X and three O markers. The player which is first in arranging all X or O array as row or column or diagonal wins the game. Figure 18.6 shows initial configuration and some moves of this game.

The game of tic-tac-toe generates all the moves at level i before generating the moves of level $i+1$, hence, it is breadth-first search (BFS). We note that while in start configuration (level 0) having all empty positions, the player MAX can make moves to level 1. The MAX will choose moves in such a way that it leads to a configuration ultimately such that all the X's are continuous and are in a line, or column or diagonal. At the same time the MAX will choose such a move that it becomes difficult for MIN later to make a move to a configuration with all O's in a line, or column or diagonal. Thus, each of MAX and MIN makes the moves so that it is winning for them and blocking for the other player.

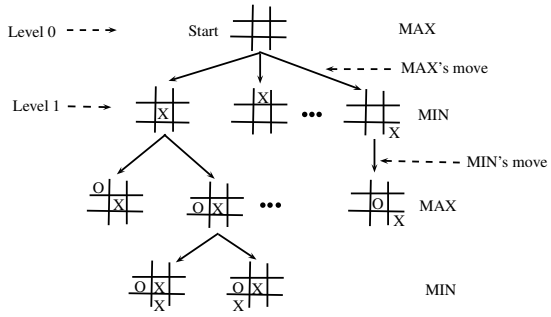


Figure 18.6: Tic-tac-toe with some initial moves.

When all the six objects (X and O) are on board, the subsequent moves by MAX or MIN are shifting of these X or O to lead them to corresponding winning configurations. To indicate which configuration is superior to move into, we associate a weight function that will be maximized by the MAX player, and the MIN player will try to minimize it. If a configuration is indicated by c, then this weight function is defined as:

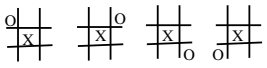
$$f(c) = m - n \tag{18.1}$$

where m = number of complete rows, columns, and diagonals that are still open (fully) for the next move of MAX, and

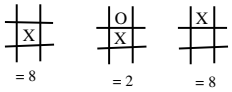
n = number of complete rows, columns, and diagonals that are till open for the next move of MIN.

If c is a winning configuration for MAX, then $f(c) = \infty$, and if c is winning configuration for MIN then $f(c) = -\infty$. If a configuration is like,

then $f(c) = 2 - 6 = -4$. We make use of symmetries in generating successor positions, then the following game states are identical.



For efficiency reasons, early in the game, the branching factor is kept small by symmetries, but late in the game, it is kept small by number of open positions available. The first move is always important. Consider that the given three initial moves for X as shown in the following configurations.



Given these configurations, if X is taking a next move, the number of rows, columns, and diagonals, put together, which are open for the next move of X are: $8, 2, 8$, respectively, for these existing configurations.

18.10 Alpha-Beta Procedure

Alpha-Beta is a tree-search procedure that is faster than minimax but still equivalent in the sense that both procedures will always choose the same depth successor at best, and will assign the same value to it. Alpha-Beta is typically several orders of magnitude faster than minimax. It saves time by not searching certain branches of the tree. Under certain conditions the values of certain branches do not affect the value which is ultimately backed up to higher levels of the tree. Hence, there is no point in evaluating these branches. When the Alpha-Beta program detects these conditions, it stops work on one branch and skips to another. This event is called an alpha or beta cutoff.

At first it appears that *static evaluator* must be used on each leaf-node at the bottom of the search-tree. But, it is not required. To see how alpha-beta search works, consider the example shown in figure 18.7. Alpha-beta starts just like minimax procedure by evaluating all the successor of P_1. The minimum of these static vales is then backed-up to P_1, since P_1 is a MIN position. The backed-up value of P_1 is *alpha* and has a value 5 in this example.

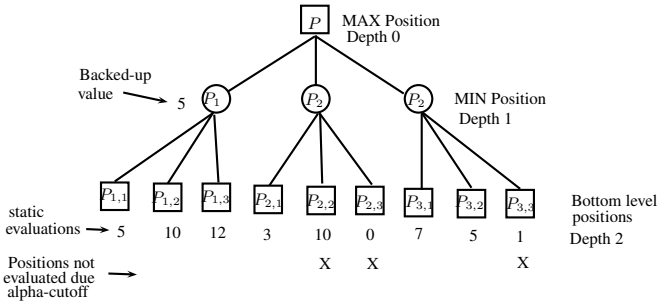


Figure 18.7: Alpha-Beta search.

Alpha is lower limit for the backed-up value of the top position, P. Since, P is max position, we back-up the value of the largest values successor of P. Since, we have evaluated only one successor at this time, we do not know that it will be 5 or larger. The value of alpha may change as the other successors are evaluated, but it can only increase, not decrease.

Having evaluated P_1, the procedure begin work on P_2. An *alpha cut-off* takes place at $P_{2,1} = 3$, as it is less than alpha. Since P_2 is a Min position, $V_{2,1}$ is an upper limit for V_2 (value of node P_2). Since V_2 is less than alpha, P_2 is definitely eliminated as a candidate for the largest valued successor of P. There is no point in evaluating the other successors of P_2 so the procedure begins work on P_3 next.

The above cut-offs saves the machine a good deal of time. The alpha cut-off at $P_{2,1}$ means that the machine need not bother to evaluate $P_{2,2}$ and $P_{2,3}$. A second alpha cut-off occurs at $P_{3,2}$, which eliminates $P_{3,3}$. Thus, in the figure 18.7 alpha-beta search program would evaluate only six of the bottom level successors, while a minimax program would evaluate all nine.

Although the example is given only for a tree of three levels, it is clear that procedure will work just the same below any Max position P_x, at any depth in a large tree, provided only that there are at least two levels below P_x. If there were more levels below $P_{1,1}$ in the example, then we could use the back-up value of $P_{1,1}$ instead of static value. If there were more levels above P, then we could backup of P.

Moreover, it is possible to pass a value of alpha down from the top of a large tree. Thus, an alpha established

at depth 1 could be used to produce cut-offs at depths 2, 4, and 6. These deep cut-offs are illustrated in figure 18.8.

Alpha is defined by the values of the successors of a Max positions (i.e., odd depths, here it is P_1, P_2, P_3 nodes, in figure 18.8), while alpha cut-off occur among the successors of a Min position (even depths, here it is $P_{1,1} \ldots P_{3,3}$, in figure 18.8). It is possible to define another variable, *Beta*, which is established at even depths and generates cut-offs at off depths. The action of beta cut-off is exactly inverse of alpha cut-offs.

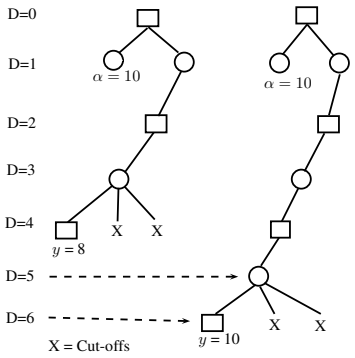


Figure 18.8: Deep Alpha cut-offs.

The effect of alpha-beta cut-off is to make the tree space grow slowly with depth. Thus, the advantage over simple MINIMAX. It is about twice as good as at Maximum depth Dmax = 3 and about thirty times as good at Dmax = 6. This is typical. It is good to have depth dependent program. Such a measure we call as depth relation DR,

$$DR = \frac{\log N}{\log N_{MM}} \tag{18.2}$$

where, N is number of nodes at the bottom of tree, and N_{MM} is number of nodes at the bottom of tree in minimax search. $DR = [0, 1]$, is effective depth in comparison to minimux procedure.

Alpha-beta search is equivalent to minimax in the sense the two procedure will choose the same depth successor as best and will always give the same value for the successor.