CS61B Lecture #23

racking searches, game trees (*DSIJ*, Section 6.5)

earching by "Generate and Test"

onsidering the problem of searching a set of data stored of data structure: "Is $x \in S$?"

we *don't* have a set S, but know how to recognize what f we find it: "Is there an x such that $P(x)$?"

ow to enumerate all possible candidates, can use approach *and Test:* test all possibilities in turn.

es be more clever: avoid trying things that won't work,

ns if the set of possible candidates is infinite?

Backtracking Search

search is one way to enumerate all possibilities.

night's Tour. Find all paths a knight can travel on a such that it touches every square exactly once and ends t move from where it started.

le below, the numbers indicate position numbers (knight

(N) is stuck; how to handle this?

6							
		5					
4	7						
	10		2				
8	3	0					
N		9		1			

General Recursive Algorithm

```
PATH a sequence of knight moves starting at ROW, COL
s all squares that have been hit already and
up one square away from ENDROW, ENDCOL. B[i][j] is
ow i and column j have been hit on PATH so far.
ue if it succeeds, else false (with no change to PATH).
ally with PATH containing the starting square, and
ng square (only) marked in B. */

th(boolean[][] b, int row, int col,
   int endRow, int endCol, List path) {
e() == 64)   return isKnightMove(row, col, endRow, endCol);
 all possible moves from (row, col)) {
c]) {
= true; // Mark the square
(new Move(r, c));
Path(b, r, c, endRow, endCol, path))  return true;
= false; // Backtrack out of the move.
ove(path.size()-1);


;
```


other Kind of Search: Best Move

problem of finding the *best* move in a two-person game.

sign a *heuristic value* to each possible move and pick *static valuation*).

ve can use a variety of heuristics. Some examples of ions:

aximal or minimal value to a won position (depending on

 black pieces − number of white pieces in checkers.

 sum of white piece values) − (weighted sum of black hess), such as queen=9, rook=5, knight=bishop=3, pawn=1.

 of pieces to strategic areas (center of board).

isleading. A move might give us more pieces, but set up response from the opponent.

 move, look at *opponent's* possible moves, use the best sults *for the opponent* as the value.

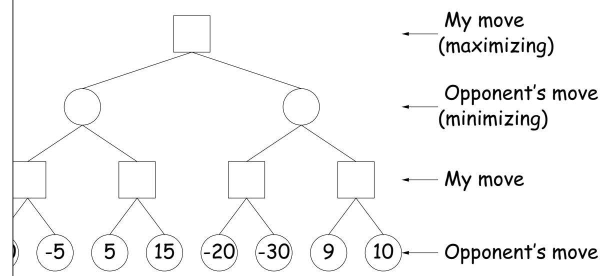
ou have a great response to opponent's response?

rganize this sensibly?

Game Trees

space of possible continuations of the game as a tree.

a position, each edge a move.

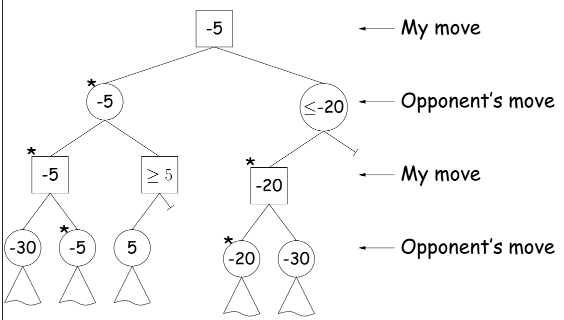


bers at the bottom are the values of those final positions er numbers are of more value to *my opponent*.

I move? What value can I get if my opponent plays as ble?

Alpha-Beta Pruning

e this tree as we search it.



position, I know that the opponent will not choose to
lready has a -5 move).

20' position, my opponent knows that I will never choose
(since I already have a -5 move).

Overall Search Algorithm

whose move it is (maximizing player or minimizing player),
for a move estimated to be optimal in one direction or

be exhaustive down to a particular depth in the game
hat, we guess values.

nd β limits:

r does not care about exploring a position further after
at its value will be larger than a position the minimizing
already found, because the minimizing player will simply
e a position with that larger value.

minimizing player won't explore a positions whose value
n what the maximizing player can get (α).

aximizing player will find a move with the call

Value(*current position*, *search depth*, $-\infty$, $+\infty$)

ayer:

Value(*current position*, *search depth*, $-\infty$, $+\infty$)

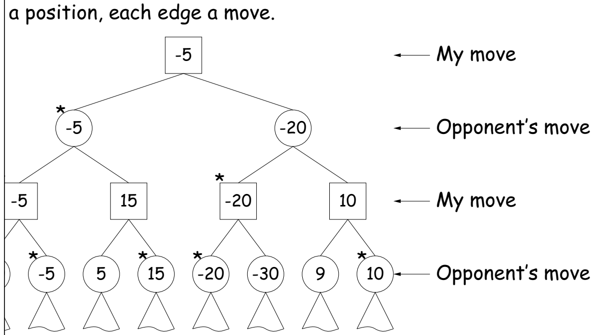
docode for Searching (Maximizing Player)

```
d minimax value of position POSN, searching up to
ahead, assuming it is the maximizing player's move.
 is determined to be <=ALPHA, then the function
ny value <=ALPHA, even if inaccurate.  Likewise if the
ETA, it may return any value >=BETA.  Assumes ALPHA<BETA. */
ue(Position posn, int depth, int alpha, int beta)

final position of the game || depth == 0)
taticGuess(posn);
r = −∞;
al move, M, in position posn) {
 next = makeMove(posn, M);
onse = minPlayerValue(next, depth-1, alpha, beta);
onse > bestSoFar) {
tSoFar = response;
ha = max(alpha, bestSoFar);
(alpha >= beta)
 return bestSoFar;


oFar;
```


Game Trees, Minimax

space of possible continuations of the game as a tree.

a position, each edge a move.



the values we guess for the positions (larger means
e). Starred nodes would be chosen.

ose child (next position) with maximum value; opponent
mum value—the *minimax algorithm*.

Cutting off the Search

traverse game tree to the bottom, you'd be able to
if it's possible).

ossible near the end of a game.

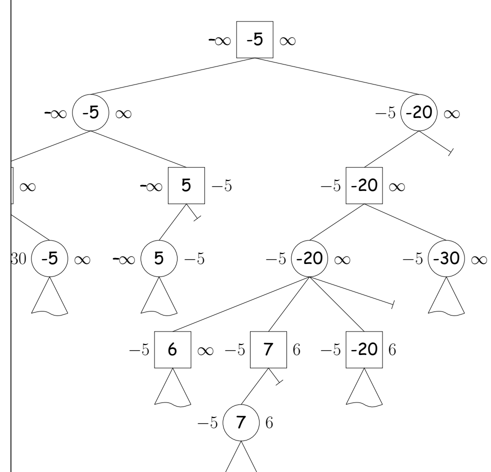
ly, game trees tend to be either infinite or impossibly

e a maximum *depth*, and use a heuristic static valuation
at that depth.

use *iterative deepening*, repeating the search at increasing
time is up.

ophisticated searches are possible, however (take CS188).

le Tree with Alpha and Beta Values



udocode for Searching (Minimizing Player)

```
d minimax value of position POSN, searching up to
ahead, assuming it is the minimizing player's move. */
ue(Position posn, int depth, int alpha, int beta)

 final position of the game || depth == 0)
taticGuess(posn);
r = +∞;
al move, M, in position posn) {
 next = makeMove(posn, M);
onse = maxPlayerValue(next, depth-1, alpha, beta);
onse < bestSoFar) {
tSoFar = response;
a = min(beta, bestSoFar);
(alpha >= beta)
 return bestSoFar;


oFar;
```