CS61B Lecture #22

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 $P(x)$?"

how to enumerate all possible candidates, can use ap- *nerate 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.

ight's Tour. Find all paths a knight can travel on a chess- hat it touches every square exactly once and ends up ove from where it started.

ole 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 evaluation*). Examples:

- black pieces – number of white pieces in checkers.
- sum of white piece values – weighted sum of black chess (Queen=9, Rook=5, etc.)
- 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, assume he st one for him, and use that as the value.

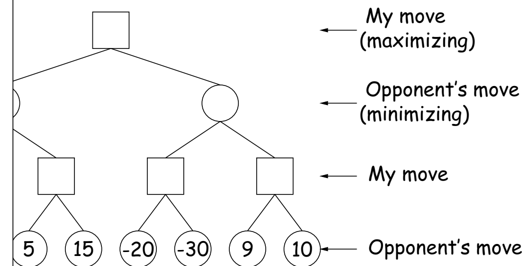
ou have a great response to his response?

rganize this sensibly?

Game Trees

space of possible continuations of the game as a tree.

a position, each edge a move.

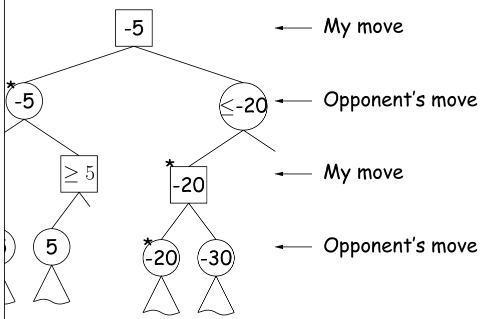


bers at the bottom are the values of those final posi- Smaller numbers are of more value to *my opponent*.

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 ince he already has a -5 move).

0' 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

e 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 once its value is larger than what the minimizing player knows (β), because the minimizing player will never allow that come about.

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

aximizing player will find a move with

urrent position, *search depth* $-\infty$, $+\infty$)

ayer:

urrent position, *search depth* $-\infty$, $+\infty$)

-Level Search for Minimizing Player

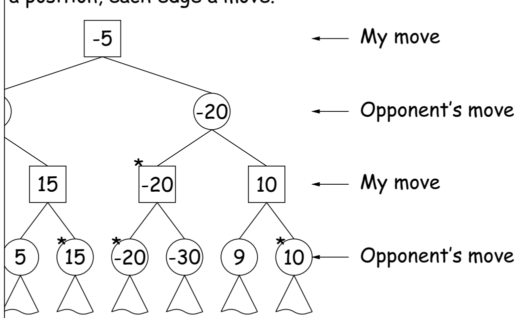
```
dMin(Position posn, double alpha, double beta) {
axPlayerWon())
 artificial "Move" with value +∞;
osn.minPlayerWon())
 artificial "Move" with value −∞;
oFar = artificial "Move" with value +∞;
M = a legal move for minimizing player from posn) {
ion next = posn.makeMove(M);
 setValue(heuristicEstimate(next));
ext.value() <= bestSoFar.value()) {
 bestSoFar = next;
 beta = min(beta, next.value());
 if (beta <= alpha) break;


tSoFar;
```


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 ("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 value computed on alone (called a *static valuation*) as the value at that

use *iterative deepening*, repeating the search at in- ths until time is up.

ophisticated searches are possible, however (take CS188).

Pseudocode for Searching (One Level)

sic kind of game-tree search is to assign some heuristic given position, looking at just the next possible move:

```
dMax(Position posn, double alpha, double beta) {
axPlayerWon())
 artificial "Move" with value +∞;
osn.minPlayerWon())
 artificial "Move" with value −∞;
oFar = artificial "Move" with value −∞;
M = a legal move for maximizing player from posn) {
ion next = posn.makeMove(M);
 setValue(heuristicEstimate(next));
ext.value() >= bestSoFar.value()) {
 bestSoFar = next;
 alpha = max(alpha, next.value());
 if (beta <= alpha) break;


tSoFar;
```

ıdocode for Searching (Minimizing Player)

```
est move for minimizing player from POSN, searching
EPTH.  Any move with value <= ALPHA is also
gh". */
osition posn, int depth, double alpha, double beta) {
 == 0 || gameOver(posn))
 simpleFindMin(posn, alpha, beta);
oFar = artificial "Move" with value +∞;
M = a legal move for minimizing player from posn) {
ion next = posn.makeMove(M);
 response = findMax(next, depth-1, alpha, beta);
esponse.value() <= bestSoFar.value()) {
 bestSoFar = next;
 next.setValue(response.value());
 beta = min(beta, response.value());
 if (beta <= alpha) break;


tSoFar;
```


ıdocode for Searching (Maximizing Player)

```
est move for maximizing player from POSN, searching
EPTH.  Any move with value >= BETA is also
gh". */
osition posn, int depth, double alpha, double beta) {
 == 0 || gameOver(posn))
 simpleFindMax(posn, alpha, beta);
oFar = artificial "Move" with value -∞;
M = a legal move for maximizing player from posn) {
ion next = posn.makeMove(M);
 response = findMin(next, depth-1, alpha, beta);
esponse.value() >= bestSoFar.value()) {
 bestSoFar = next;
 next.setValue(response.value());
 alpha = max(alpha, response.value());
 if (beta <= alpha) break;


tSoFar;
```