Today

See Russell and Norvig, chapters 5 and 6

- Local search for CSPs
- 3SAT
- Adversarial Search

Reminder: Constraint satisfaction problems

CSP:
state is defined by *variables* X_i with *values* from *domain* D_i
goal test is a set of *constraints* specifying
allowable combinations of values for subsets of variables

Simple example of a *formal representation language*

Allows useful *general-purpose* algorithms with more power
than standard search algorithms

Iterative algorithms for CSPs

Hill-climbing typically works with "complete" states, i.e., all variables assigned

To apply to CSPs:
allow states with unsatisfied constraints
operators **reassign** variable values

Variable selection: randomly select any conflicted variable

Value selection by **min-conflicts** heuristic:
choose value that violates the fewest constraints
i.e., hillclimb with $h(n) =$ total number of violated constraints

A standard CSP problem

A famous and much studied problem is known as **3SAT**. This is a Boolean CSP (i.e. the variables take the values **true,false**).

Each constraint here is of the form

$$(\neg)V_i \vee (\neg)V_j \vee (\neg)V_k$$

where each variable may be negated. For example, the constraint $A \vee B \vee \neg C$ says that either A is true, or B is true or C is false.

Solving such a constraint problem over n variables is hard.

The only known algorithms for this are exponential in n.

However, we have no proof that there is no polynomial algorithm.

If you find a poly algorithm, you will be famous!!

Iterative algorithms for 3SAT

Iterative methods are often used for 3SAT. Start with a random assignment of true/false to variables, and flip values to try to remove conflicts.

A recent favoured algorithm is called $WALKSAT$:

`www.cs.rochester.edu/u/kautz/walksat`

The algorithm is simple.

WALKSAT

Basic algorithm; try repeatedly from different initial assignment; parametrised by `MAX-TRIES` and number of repeated attempts

```
Procedure GSAT
FOR i:= 1 to MAX-TRIES
  T := random truth assignment
  FOR j:= 1 to MAX-FLIPS
      IF T satisfies Constraints then return T
      Flip any variable that gives greatest increase
         in number of satisfied constraints (can be 0,negative)
  end FOR
end FOR
return Failure
```


WALKSAT ctd

- can escape from local maxima (allows "negative" moves")
- restarting also helps; best to use both possibilities
- this is still incomplete in general
- local search is surprisingly good for problems like 3SAT; can deal with problems with thousands of variables and clauses.

Games vs. search problems

"Unpredictable" opponent ⇒ solution is a strategy specifying a move for every possible opponent reply

Time limits ⇒ unlikely to find goal, must approximate

Plan of attack:

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)

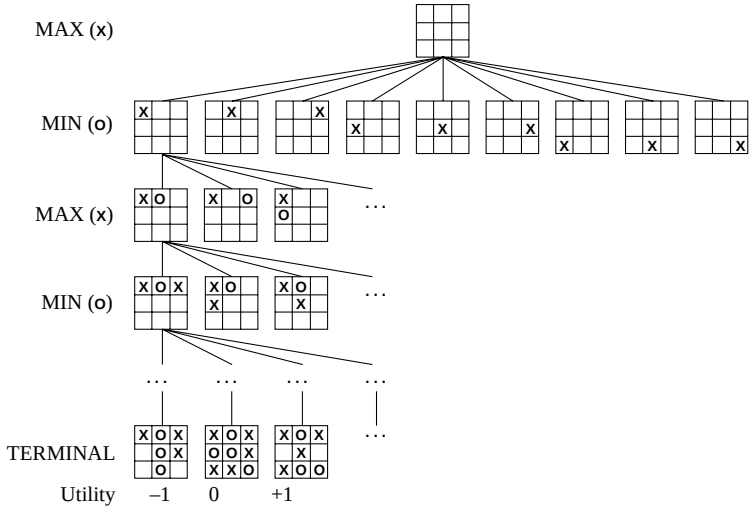
Types of games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war

Game tree (2-player, deterministic, turns)

Example for noughts and crosses (tictactoe).

- Alternate layers in the tree correspond to the different players
- Both players know all about the current state of the game
- Each leaf in the tree represents win for one player (or draw)

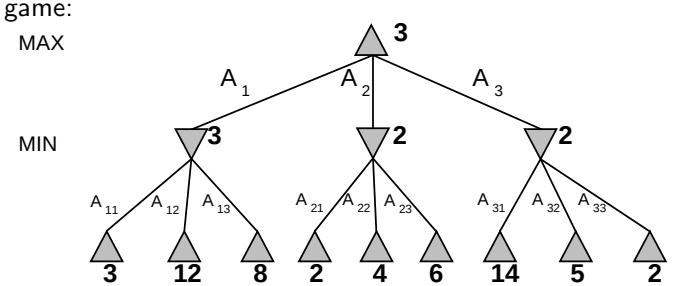


Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest *minimax value*
= best achievable payoff against best play

E.g., 2-ply game:



Minimax algorithm

function MINIMAX-DECISION(*state*, *game*) **returns** *an action*

action, *state* ← the *a*, *s* **in** SUCCESSORS(*state*)
such that MINIMAX-VALUE(*s*, *game*) is maximized
return *action*

---

function MINIMAX-VALUE(*state*, *game*) **returns** *a utility value*

if TERMINAL-TEST(*state*) **then**
return UTILITY(*state*)
else if MAX is to move in *state* **then**
return the highest MINIMAX-VALUE of SUCCESSORS(*state*)
else
return the lowest MINIMAX-VALUE of SUCCESSORS(*state*)

Properties of minimax

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity?? $O(b^m)$

Space complexity?? $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
$\Rightarrow$ exact solution completely infeasible

Resource limits

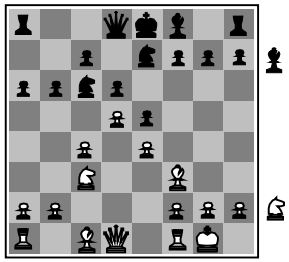
Suppose we have 100 seconds, explore 10^4 nodes/second
$\Rightarrow 10^6$ nodes per move

Standard approach:

- *cutoff test*
  e.g., depth limit (perhaps add *quiescence search*)

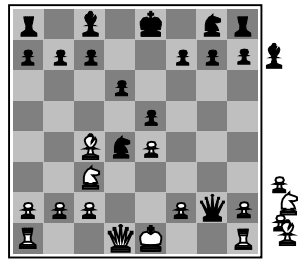
- *evaluation function*
  = estimated desirability of position

Evaluation functions



Black to move

White slightly better

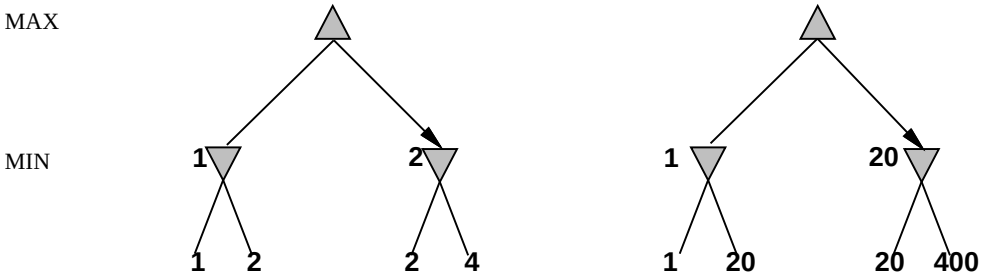


White to move

Black winning

For chess, typically *linear* weighted sum of features
$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
e.g., $w_1 = 9$ with $f_1(s) =$ (number of white queens) – (number of black queens)

Digression: Exact values don't matter



Behaviour is preserved under any *monotonic* transformation of Eval

Only the order matters:
payoff in deterministic games acts as an *ordinal utility* function

Cutting off search

MinimaxCutoff is identical to MinimaxValue except
1. Terminal? is replaced by Cutoff?
2. Utility is replaced by Eval

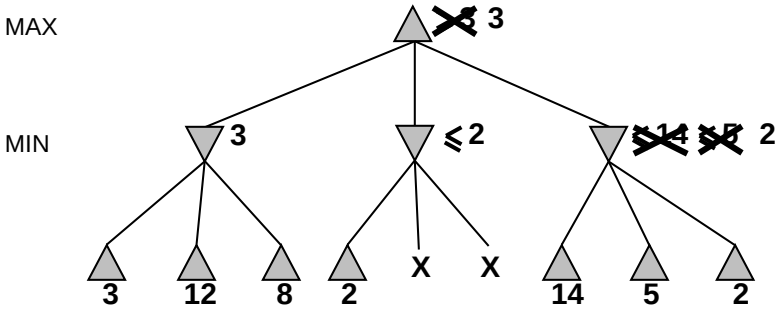
Does it work in practice?

$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

4-ply lookahead is a hopeless chess player!

4-ply ≈ human novice
8-ply ≈ typical PC, human master
12-ply ≈ Deep Blue, Kasparov

α–β pruning example



Properties of α–β

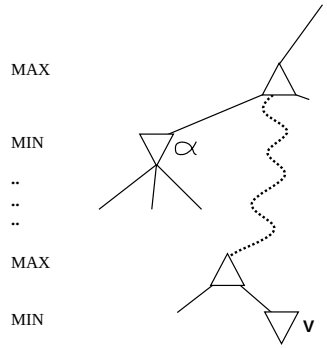
Pruning *does not* affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity $= O(b^{m/2})$
⇒ *doubles* depth of search
⇒ can easily reach depth 8 and play good chess

A simple example of the value of reasoning about which computations are relevant (a form of *metareasoning*)

Why is it called α–β?



α is the best value (to MAX) found so far off the current path; if V is worse than α, MAX will avoid it $\Rightarrow$ prune that branch. Define β similarly for MIN

The α–β algorithm

function ALPHA-BETA-SEARCH($state, game$) **returns** an action
  $action$, $state \leftarrow$ the a, s **in** SUCCESSORS[$game$]($state$)
    such that MIN-VALUE($s, game, -\infty, +\infty$) is maximized
  return $action$

The α–β algorithm ctd.

function MAX-VALUE($state, game, \alpha, \beta$) **returns** the minimax value of $state$
  if CUTOFF-TEST($state$) **then return** EVAL($state$)
  for each s **in** SUCCESSORS($state$) **do**
    $\alpha \leftarrow \max(\alpha,$ MIN-VALUE($s, game, \alpha, \beta$))
    if $\alpha \geq \beta$ **then return** β
  return α

function MIN-VALUE($state, game, \alpha, \beta$) **returns** the minimax value of $state$
  if CUTOFF-TEST($state$) **then return** EVAL($state$)
  for each s **in** SUCCESSORS($state$) **do**
    $\beta \leftarrow \min(\beta,$ MAX-VALUE($s, game, \alpha, \beta$))
    if $\beta \leq \alpha$ **then return** α
  return β

Deterministic games in practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.

Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

Summary

- Local search for CSPs
- Adversarial search
- Search in games with perfect information