Gradient Descent/Ascent

“Like climbing Everest in thick fog with amnesia”

```
function HILL-CLIMBING(problem) returns a solution state
    inputs: problem, a problem
    local variables: current, a node
                    next, a node

    current ← Make-Node(Initial-State[problem])
    loop do
        next ← a highest-valued successor of current
        if VALUE[next] < VALUE[current] then return current
        current ← next
    end
```
Simulated Annealing

Idea: escape local maxima by allowing some “bad” moves but gradually decrease their size and frequency

```plaintext
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
             schedule, a mapping from time to “temperature”
    local variables: current, a node
                     next, a node
                     T, a “temperature” controlling the probability of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T=0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] - VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^{ΔE/T}
```
Simulated Annealing Properties

At fixed "temperature" $T$, state occupation probability reaches Boltzmann distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

$T$ decreased slowly enough $\implies$ always reach best state

*Is this necessarily an interesting guarantee??*

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.
What’s in a Proof? Proof of Asymptotic Convergence

since \( \lim_{x \downarrow 0} a^{b/x} = 1 \) if \( b = 0 \) and \( 0 \) if \( b < 0 \)

\[
\lim_{c \downarrow 0} P(X = i) = \lim_{c \downarrow 0} \exp \left( \frac{-f(i)}{c} \right)
\sum_{j \in S} \exp \left( \frac{-f(j)}{c} \right)
= \lim_{c \downarrow 0} \exp \left( \frac{f_{opt} - f(i)}{c} \right)
\sum_{j \in S} \exp \left( \frac{f_{opt} - f(j)}{c} \right)
= \lim_{c \downarrow 0} \frac{1}{\sum_{j \in S} \exp \left( \frac{f_{opt} - f(j)}{c} \right)} \chi(S_{opt})(i)
+ \lim_{c \downarrow 0} \exp \left( \frac{f_{opt} - f(i)}{c} \right)
\sum_{j \in S \setminus S_{opt}} \exp \left( \frac{f_{opt} - f(j)}{c} \right) \chi(S \setminus S_{opt})(i)
= \frac{1}{|S_{opt}|} \chi(S_{opt})(i) + 0
\]

\[\text{hence:} \]
\[\lim_{c \downarrow 0} P(X \in S_{opt}) = 1\]
Genetic Algorithms - Briefly

- invented in mid-1970s by John Holland

- GAs used the principle of natural selection to solve complicated optimization problems
  - transportation scheduling
  - car dealership location
  - pipeline sizing
  - VLSI chip layout

- However GAs are not a panacea ...
  - their efficiency is critically dependant on finding a good representation and good operators
An Overview of GA’s

A genetic algorithm is a computer simulation in which a population of abstract representations of candidate solutions to an optimization problem are stochastically selected, recombined, mutated, and then either eliminated or retained, based on their relative fitness.

- from an initial population
- select individuals
- breed them
- mutate the offspring
- insert them into the next generation
Basic GA Representation

the original Holland's GA is used as a starting point for almost all new work

a simple GA represents individuals solutions using string of bits

```
1 0 1 1 0 1 0 0 1 1 1 1 0
```

this bits may encode integers, real numbers, sets, or whatever ...

original analysis of GAs assumed binary representation

makes operators simple

most practical GAs use problem-specific representation
Simple Cross Over

- 1-point cross-over

```
father: [representation of chromosome]
mother: [representation of chromosome]
child: [representation of chromosome]
```

- Several different forms of crossover are now used, and inversion is no more use.

- Mutation
Creation of the Next Generation

- selection based directly on fitness

- generational update
  - create entirely a new population and replace all of the old one
  - no breeding possible between individuals from different generations

- single population
  - anyone could potentially breed with any other

Representation, cross-over, mutation, fitness for n-Queens problem, TSP

http://www-cse.uta.edu/~cook/ai1/lectures/applets/gatsp/TSP.html
Adversarial Search

• Basic minimax search
• Evaluation Functions (resource limited)
• Alpha beta pruning
• Techniques used in board playing
  – World champion of Checkers, Othello is a machine (http://www.cs.ualberta.ca/~chinook/)
  – Machines challenging in Chess, Backgammon
Games and Search

“Unpredictable” opponent ⇒ solution is a contingency plan

Time limits ⇒ unlikely to find goal, must approximate

Plan of attack:

• algorithm for perfect play (Von Neumann, 1944)
• finite horizon, approximate evaluation (Zuse, 1945; Shannon, 1950; Samuel, 1952–57)
• pruning to reduce costs (McCarthy, 1956)

<table>
<thead>
<tr>
<th>perfect information</th>
<th>deterministic</th>
<th>chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>chess, checkers, go, othello</td>
<td>backgammon monopoly</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>imperfect information</th>
<th>deterministic</th>
<th>chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>???</td>
<td>bridge, poker, scrabble nuclear war</td>
<td></td>
</tr>
</tbody>
</table>
Game Trees and 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:

```
function Minimax-Decision(game) returns an operator
    for each op in Operators[game] do
        Value[op] ← Minimax-Value(Apply(op, game), game)
    end
    return the op with the highest Value[op]

function Minimax-Value(state, game) returns a utility value
    if Terminal-Test(game)(state) then
        return Utility[game](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)
```
Minimax Example
Properties of Minimax Search

Complete??
Optimal??
Time complexity??
Space complexity??
Properties of Minimax Search

**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
Three Player Games

How do we create game trees that incorporate chance?
Chance as a Bipartisan Third Player

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:

\[
\begin{array}{c}
\text{MAX} \\
\text{CHANCE} \\
\text{MIN}
\end{array}
\]

\[
\begin{array}{c}
\text{Result} \\
2 \\
4 \\
0 \\
-2
\end{array}
\]

\[
\begin{array}{cc}
0.5 & 0.5 \\
0.5 & 0.5
\end{array}
\]

\[
\begin{array}{cc}
3 & -1
\end{array}
\]

**Expectiminimax** gives perfect play

Just like **Minimax**, except we must also handle chance nodes:

\[
\ldots
\]

if `state` is a **Max** node then

\[
\text{return the highest } \text{Expectiminimax-Value of Successors}(state)
\]

if `state` is a **Min** node then

\[
\text{return the lowest } \text{Expectiminimax-Value of Successors}(state)
\]

if `state` is a chance node then

\[
\text{return average of } \text{Expectiminimax-Value of Successors}(state)
\]

\[
\ldots
\]
Depth Limited Search

Minimax Cutoff is identical to Minimax Value except
1. Terminal? is replaced by Cutoff?
2. Utility is replaced by Eval

Evaluation function is just a heuristic (not an admissible heuristic)
Break game state into m components
Eval(n) = w₁c₁(n) + w₂c₂(n) ... wₘcₘ(n)
Note multiple states have the exact same component values
Eval function some estimate of the the expectation of #wins/#games from this point (as described by the components)
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

$$ f_1(s) = (\text{number of white queens}) - (\text{number of black queens}) $$

etc.
Example – 3D Tic-Tac-Toe

Depth of tree?
Number of nodes?
Evaluation function?
What are the components?
How should we score each component?
Close Enough is Good Enough

Behaviour is preserved under any *monotonic* transformation of $E_{\text{VAL}}$

Only the order matters:
- payoff in deterministic games acts as an *ordinal utility* function
Cutting of 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 \( \approx \) human novice
8-ply \( \approx \) typical PC, human master
12-ply \( \approx \) Deep Blue, Kasparov
Some branches will never be played by rational players since they include sub-optimal decisions (for either player).

What is vital to the success of this approach?
Result

nodes that were never explored !!!
Alpha-Beta pruning

function Max-Value(state, game, \(\alpha\), \(\beta\)) returns the minimax value of state
inputs: state, current state in game
        game, game description
        \(\alpha\), the best score for MAX along the path to state
        \(\beta\), the best score for MIN along the path to state

if GOAL-Test(state) then return EVAL(state)
for each s in Successors(state) do
    \(\alpha\) \leftarrow \text{Max}(\alpha, \text{Min-Value}(s, game, \alpha, \beta))
    if \(\alpha \geq \beta\) then return \(\beta\)
end
return \(\alpha\)

function Min-Value(state, game, \(\alpha\), \(\beta\)) returns the minimax value of state

if GOAL-Test(state) then return EVAL(state)
for each s in Successors(state) do
    \(\beta\) \leftarrow \text{Min}(\beta, \text{Max-Value}(s, game, \alpha, \beta))
    if \(\beta \leq \alpha\) then return \(\alpha\)
end
return \(\beta\)
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.


Othello: human champions refuse to compete against computers, who are too good.

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.