Artificial Intelligence

(Please notice that this document has been combined from several sources. Some of the material might be irrelevant).

Textbooks

- Artificial Intelligence : Structures and Strategies for Complex Problem Solving
  George Luger and William Stubblefield
  Benjamin/ Cummings
- Artificial Intelligence : A modern Approach
  Stuart Russell and Peter Norvig
  Prentice-Hall
- Machine Learning
  Tom Mitchell
  McGraw-Hill
- www.aaai.org/AI Topics

What is Intelligence?

Intelligence is an interior characteristic. Its presence can not be measured directly. It can be related to :

- perception and comprehension
- making generalizations and relationships
- solving complex problems (labyrinth traversal – monkey + bananas + boxes in a room - language learning – talking …)

In 1976, Newell and Simon proposed that intelligence resides in physical symbol systems (collection of patterns and processes).

What is Artificial Intelligence ?

- Cognitive AI (Study of mind structure and its processes)
  - Study of mental faculties (seeing, learning, remembering, and reasoning) through computational models

- Engineering AI
  - Making computers do what people currently do better
  - Study of heuristic solutions to NP-complete problems

Ancestors of AI (Multidisciplinary Science)

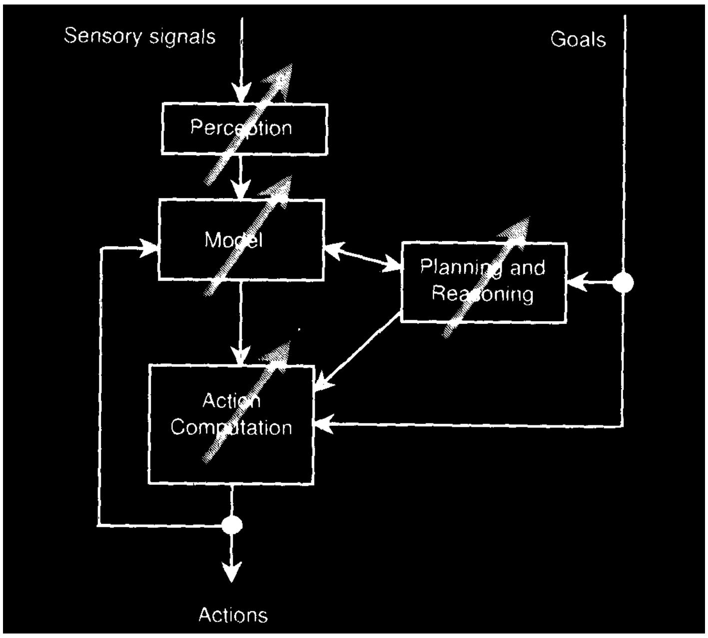
- Computer Science
- Mathematics
- Philosophy

- Probability and statistics
- Decision theory and econonmies
- Psychology
- Biology
- Control systems
- Operations research

This gives us four possible goals to pursue in artificial intelligence:

Systems that think like humans.	Systems that think rationally.
Systems that act like humans	Systems that act rationally

Acting humanly: The Turing Test approach
Thinking humanly: The cognitive modelling approach
Thinking rationally: The laws of thought approach
Thinking rationaly = to obtain correct con clusions given correct premises.
Acting rationally: The rational agent approach



Architecture of an AI System (Agent)

(If changes can be made to any functional unit - as indicated by the arrows- this implies that the system can adapt or learn).

Historical Perspective

- formalizing the laws of human thought

(4th C BC+) Aristotle, George Boole, Gottlob Frege, Alfred Tarski

- formalizing probabilistic reasoning

(16th C+) Gerolamo Cardano, Pierre Femat, James Bernoulli, Thomas Bayes

- thinking as computation

(1950+) Alan Turing, John von Neumann, Claude Shannon

- start of the field of AI

(1956) John McCarthy, Marvin Minsky, Herbert Simon, Allen Newell

AI has gone through 3 phases

- General Problem Solving : 50's
- Expert Sytems : 70's
- Machine Learning ; 80's

Problem solving and search

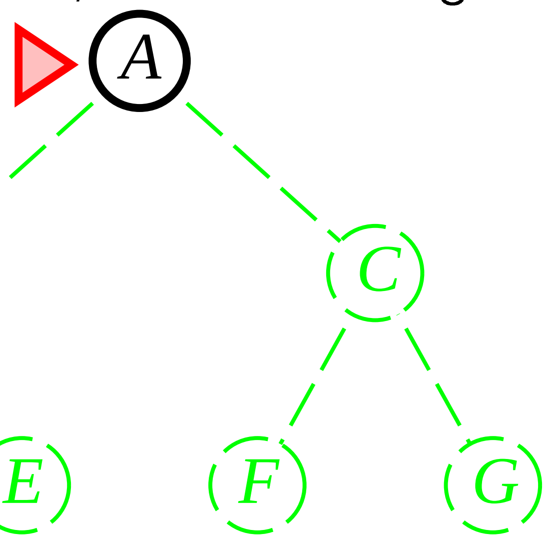
Chapter 3, Sections 1–5

Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

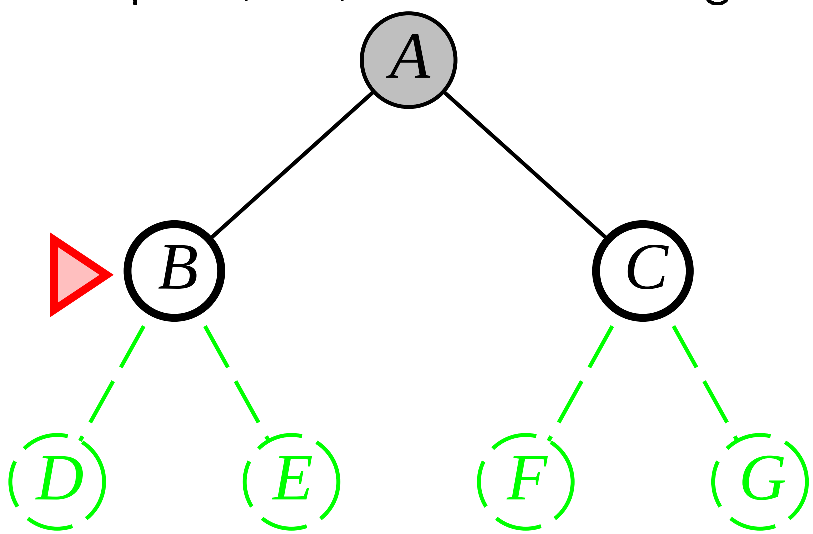


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

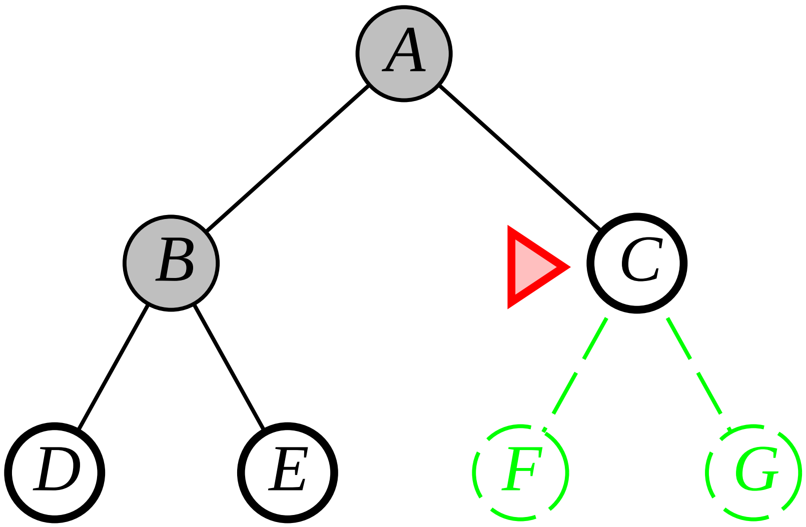


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end

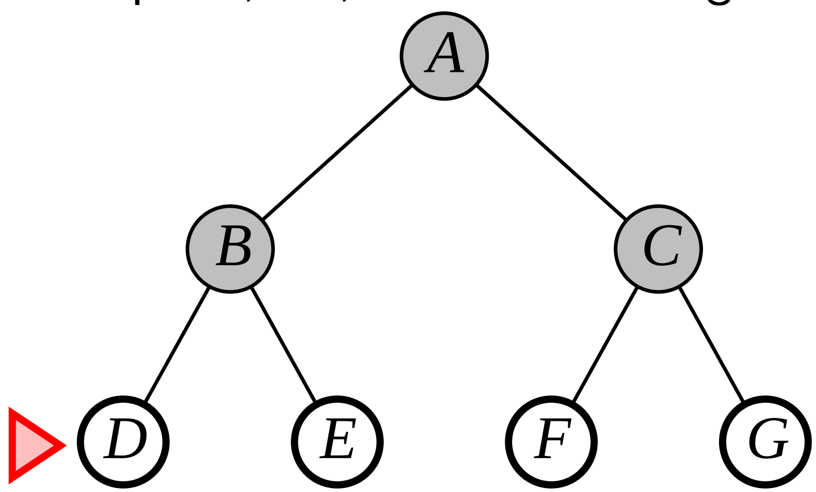


Breadth-first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end



Properties of breadth-first search

Complete??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space?? $O(b^{d+1})$ (keeps every node in memory)

Optimal??

Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space?? $O(b^{d+1})$ (keeps every node in memory)

Optimal?? Yes (if cost $= 1$ per step); not optimal in general

Space is the big problem; can easily generate nodes at 10MB/sec
so 24hrs = 860GB.

Uniform-cost search

Expand least-cost unexpanded node

Implementation:

$fringe =$ queue ordered by path cost

Equivalent to breadth-first if step costs all equal

Complete?? Yes, if step cost $\geq \epsilon$

Time?? $\#$ of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$
where C^* is the cost of the optimal solution

Space?? $\#$ of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$

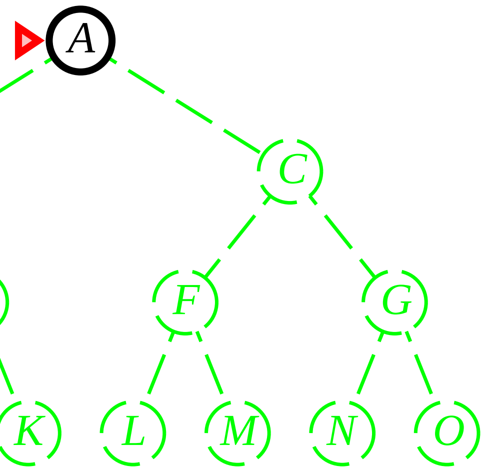
Optimal?? Yes—nodes expanded in increasing order of $g(n)$

Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

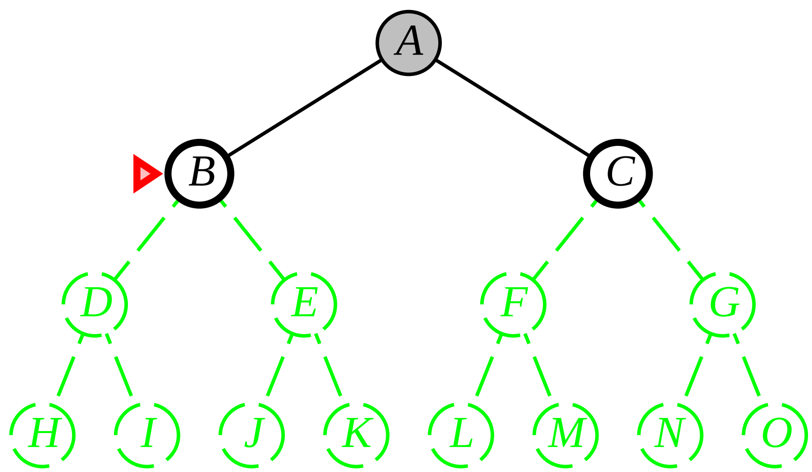


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

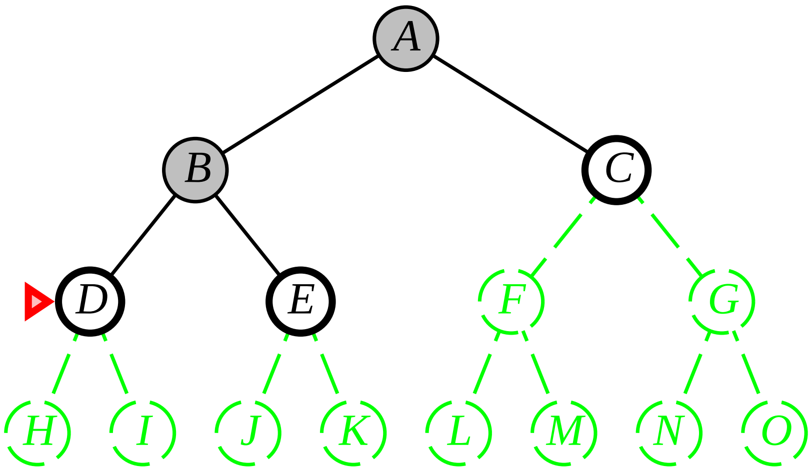


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

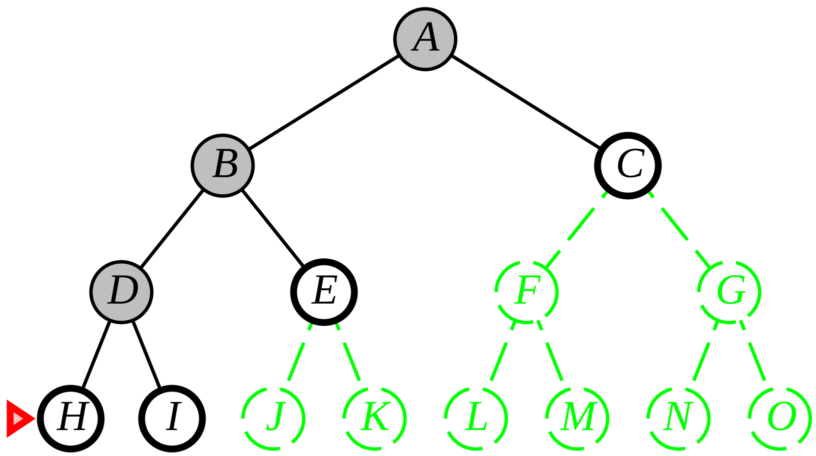


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

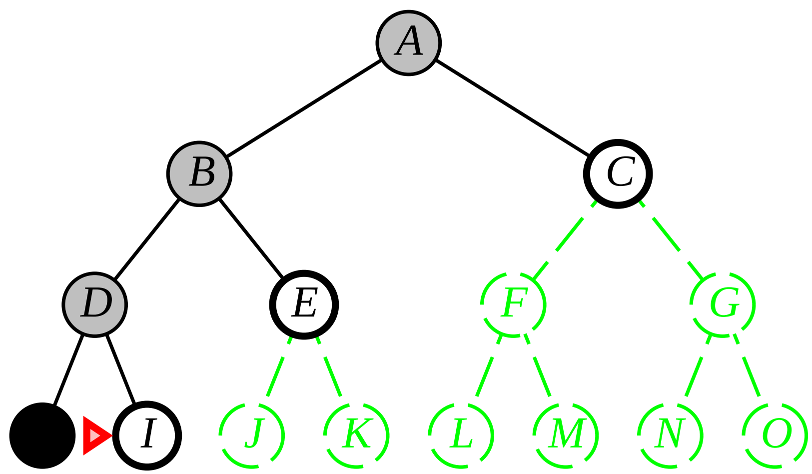


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe =$ LIFO queue, i.e., put successors at front

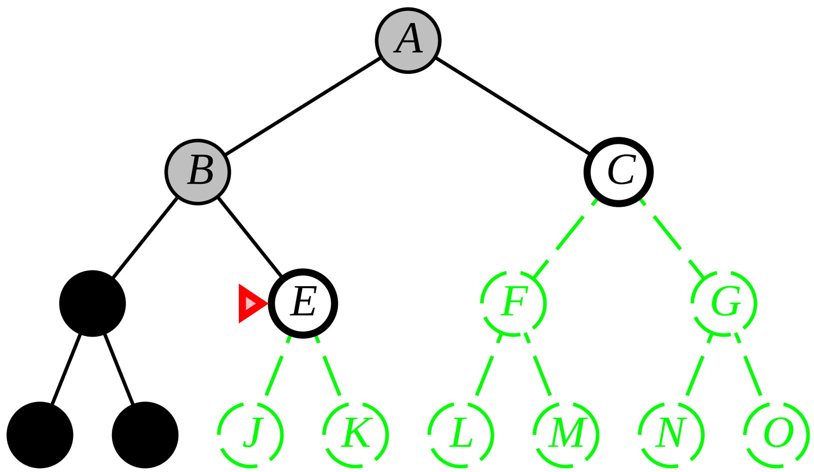


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

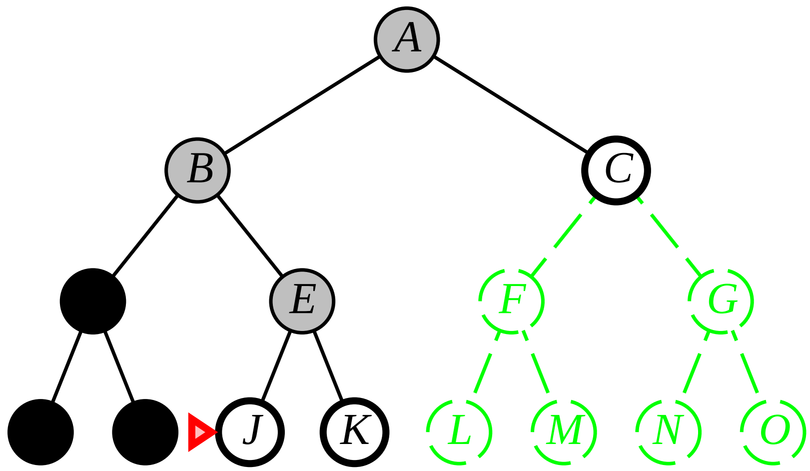


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

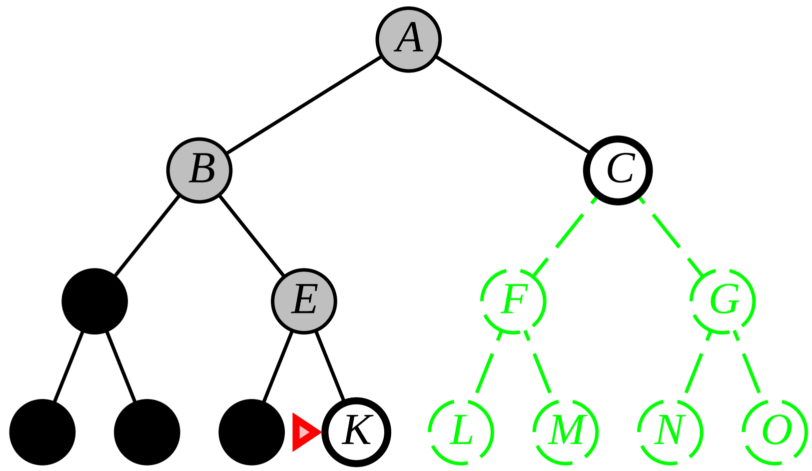


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe =$ LIFO queue, i.e., put successors at front

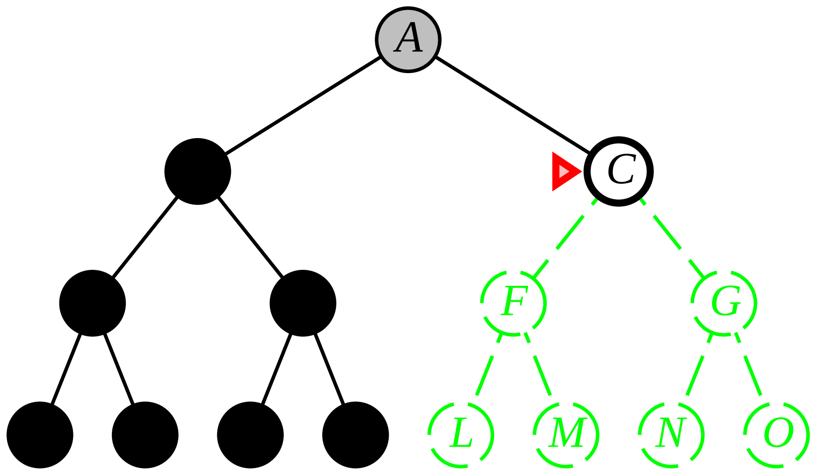


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

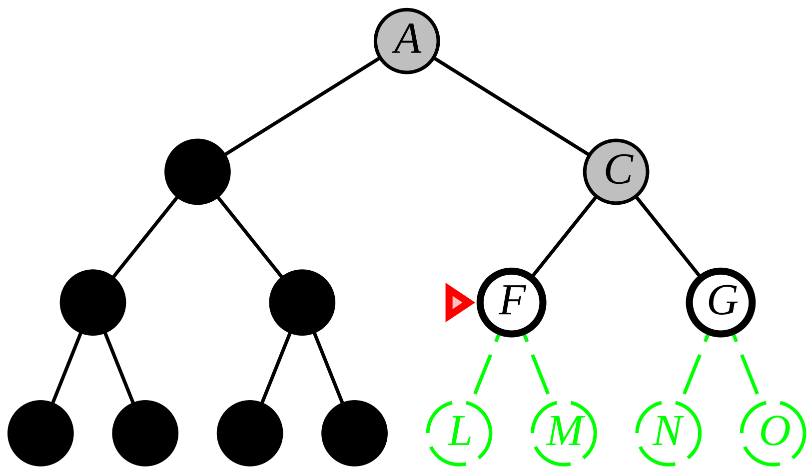


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe$ = LIFO queue, i.e., put successors at front

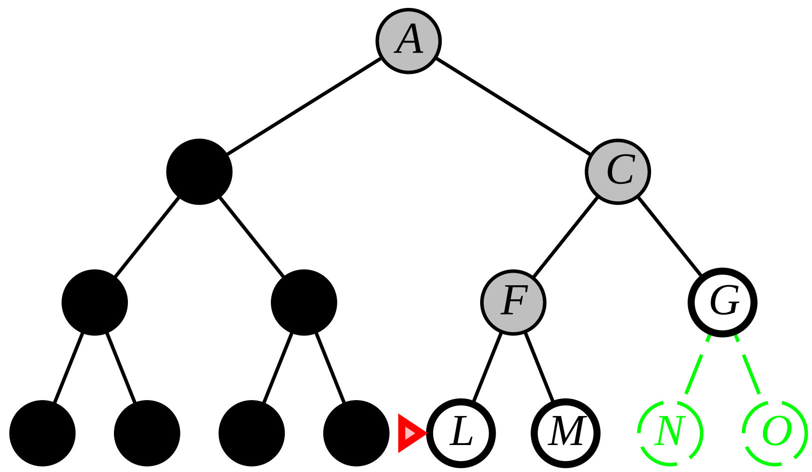


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe =$ LIFO queue, i.e., put successors at front

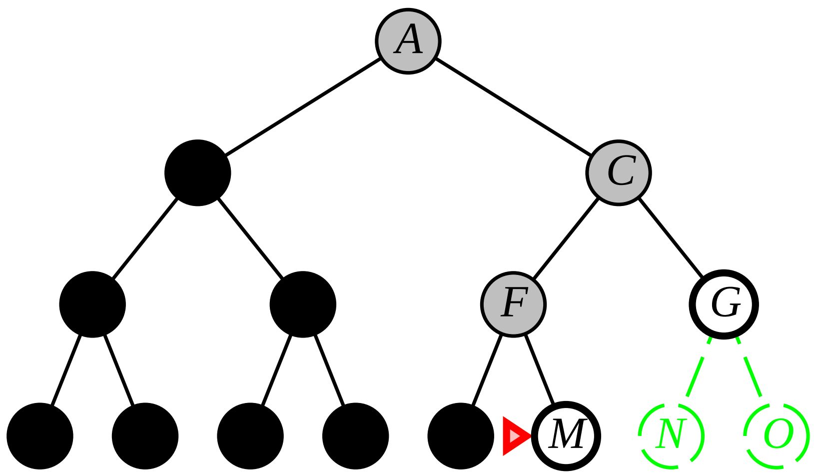


Depth-first search

Expand deepest unexpanded node

Implementation:

$fringe =$ LIFO queue, i.e., put successors at front



Properties of depth-first search

Complete??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops

Modify to avoid repeated states along path

$\Rightarrow$ complete in finite spaces

Time??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first

Space??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first

Space?? $O(bm)$, i.e., linear space!

Optimal??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first

Space?? $O(bm)$, i.e., linear space!

Optimal?? No

Depth-limited search

= depth-first search with depth limit l,
i.e., nodes at depth l have no successors

Recursive implementation:

function DEPTH-LIMITED-SEARCH(*problem*, *limit*) **returns** soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[*problem*]), *problem*, *limit*)

function RECURSIVE-DLS(*node*, *problem*, *limit*) **returns** soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST[*problem*](STATE[*node*]) **then return** *node*
    else if DEPTH[*node*] = *limit* **then return** *cutoff*
    else for each *successor* **in** EXPAND(*node*, *problem*) **do**
        result ← RECURSIVE-DLS(*successor*, *problem*, *limit*)
        if *result* = *cutoff* **then** *cutoff-occurred?* ← true
        else if *result* ≠ *failure* **then return** *result*
    if *cutoff-occurred?* **then return** *cutoff* **else return** *failure*

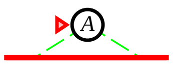
Iterative deepening search

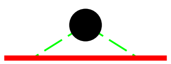
function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution
    inputs: *problem*, a problem

    for *depth* ← 0 **to** ∞ **do**
        result ← DEPTH-LIMITED-SEARCH(*problem*, *depth*)
        if *result* ≠ cutoff **then return** *result*
    end

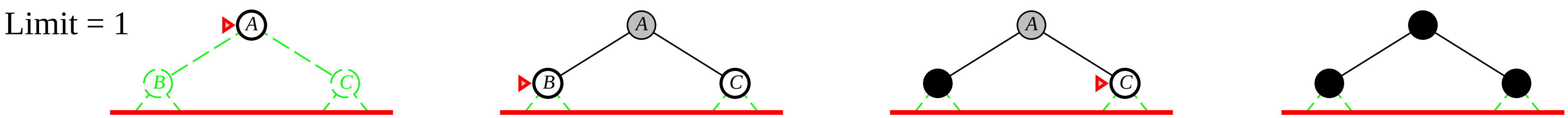
Iterative deepening search $l = 0$

Limit = 0

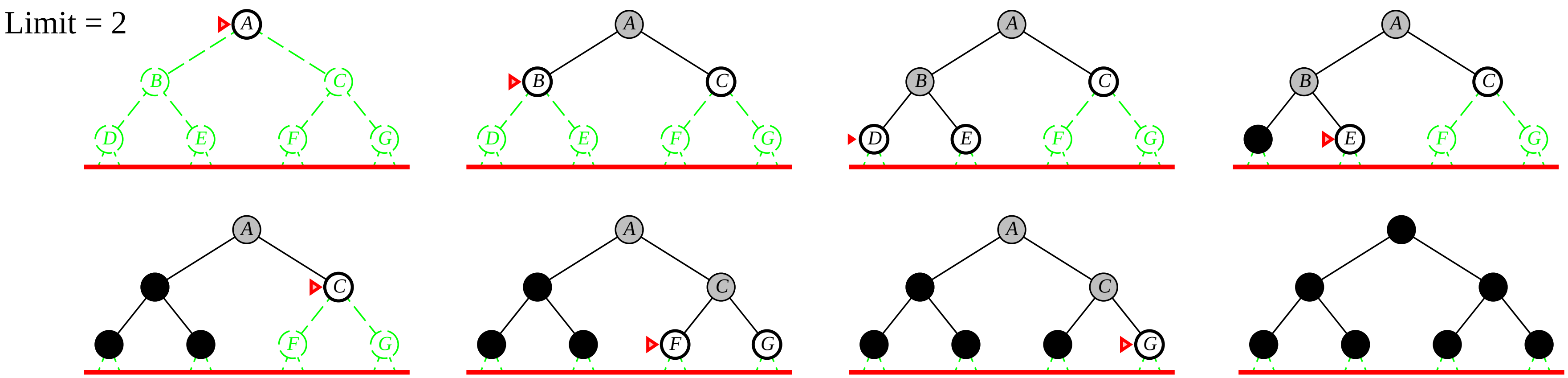




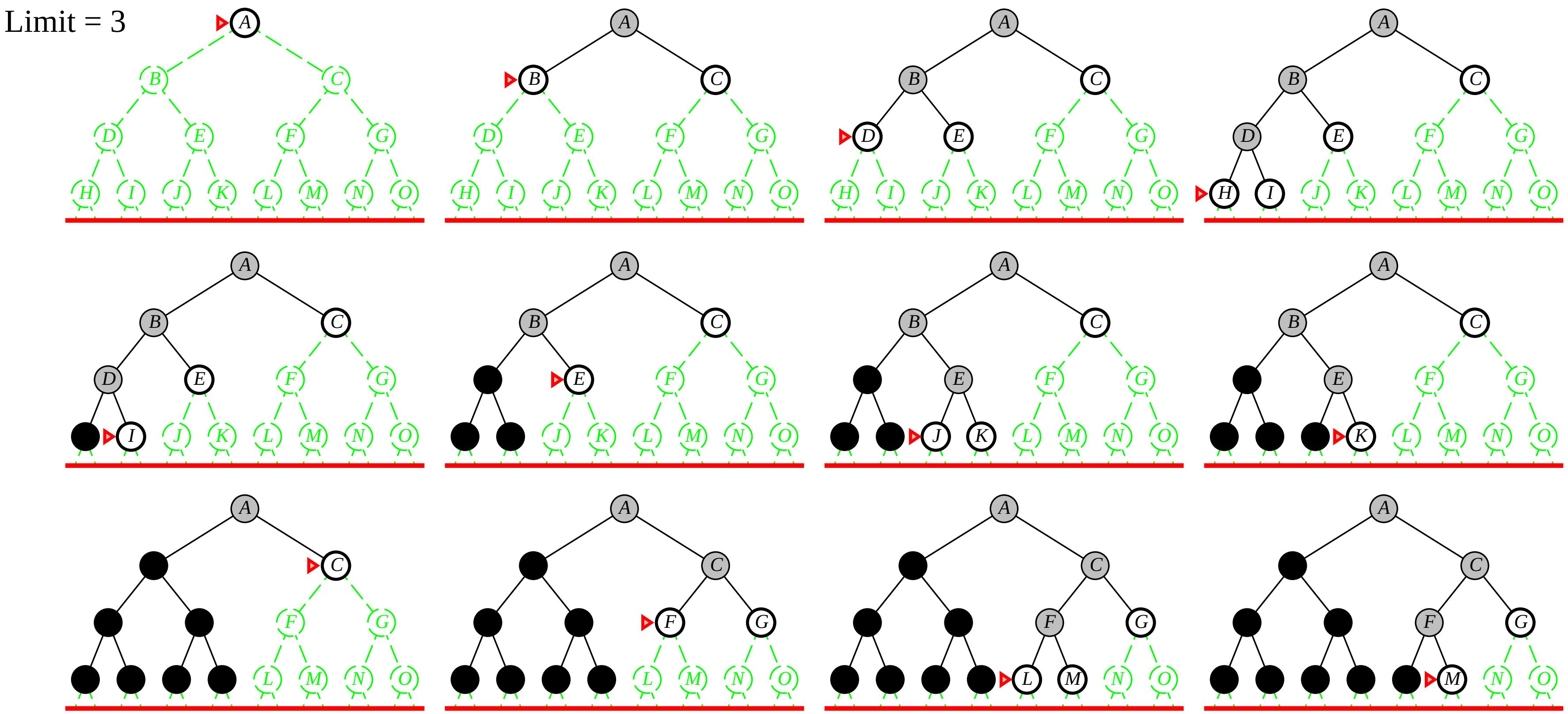
Iterative deepening search $l = 1$



Iterative deepening search $l = 2$



Iterative deepening search $l = 3$



Properties of iterative deepening search

Complete??

Properties of iterative deepening search

Complete?? Yes

Time??

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space??

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal??

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? Yes, if step cost $= 1$

Can be modified to explore uniform-cost tree

Numerical comparison for $b = 10$ and $d = 5$, solution at far right:

$$N(\mathsf{IDS}) \ = \ 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$
$$N(\mathsf{BFS}) \ = \ 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100$$

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes*	No	No	Yes

Graph search

```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure

   closed ← an empty set
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return node
        if STATE[node] is not in closed then
             add STATE[node] to closed
             fringe ← INSERTALL(EXPAND(node, problem), fringe)
   end
```

Summary

Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored

Variety of uninformed search strategies

Iterative deepening search uses only linear space
and not much more time than other uninformed algorithms

Informed search algorithms

Chapter 4, Sections 1–2, 4

Outline

◇ Best-first search

◇ A^* search

◇ Heuristics

◇ Hill-climbing

◇ Simulated annealing

Review: Tree search

function TREE-SEARCH(*problem*, *fringe*) **returns** a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[*problem*]), *fringe*)
  loop do
    if *fringe* is empty **then return** failure
    node ← REMOVE-FRONT(*fringe*)
    if GOAL-TEST[*problem*] applied to STATE(*node*) succeeds **return** *node*
    fringe ← INSERTALL(EXPAND(*node*, *problem*), *fringe*)

A strategy is defined by picking the *order of node expansion*

Best-first search

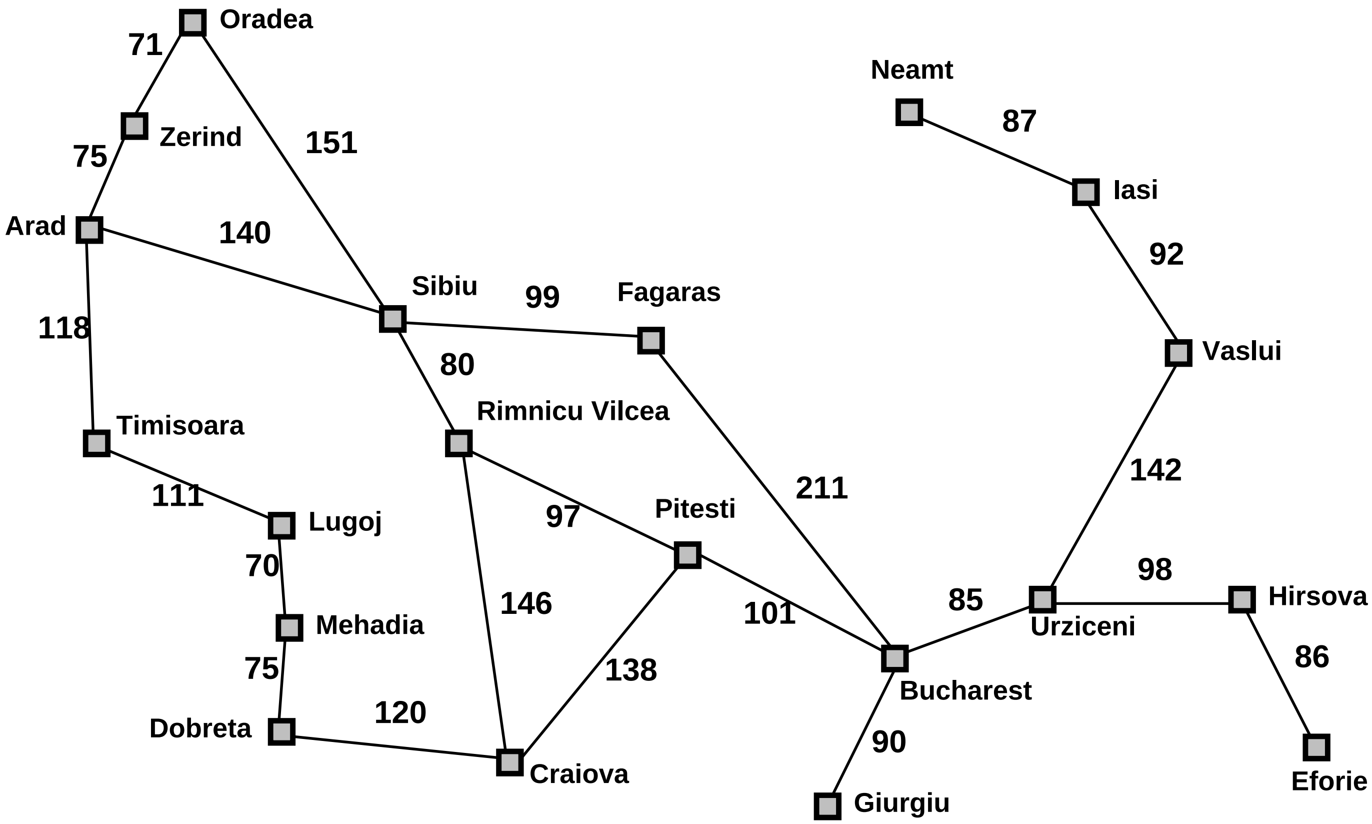
Idea: use an *evaluation function* for each node
  – estimate of "desirability"

$\Rightarrow$ Expand most desirable unexpanded node

Implementation:
$fringe$ is a queue sorted in decreasing order of desirability

Special cases:
  greedy search
  A* search

Romania with step costs in km



Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search

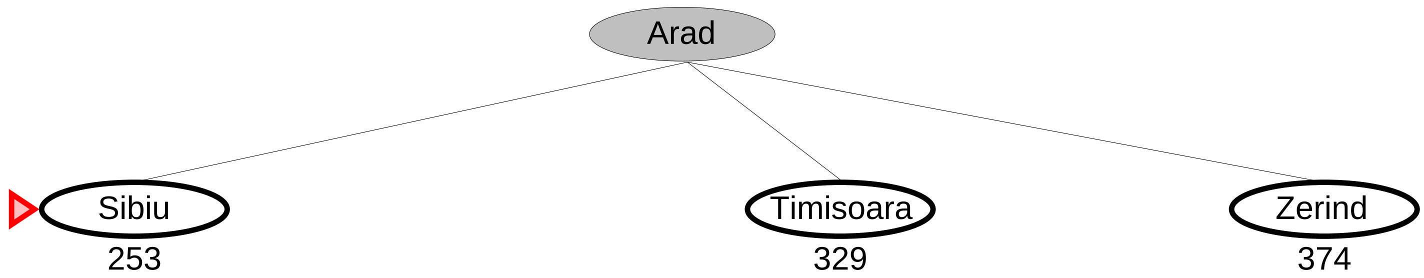
Evaluation function $h(n)$ (heuristic)
$= $ estimate of cost from n to the closest goal

E.g., $h_{\mathrm{SLD}}(n) =$ straight-line distance from n to Bucharest

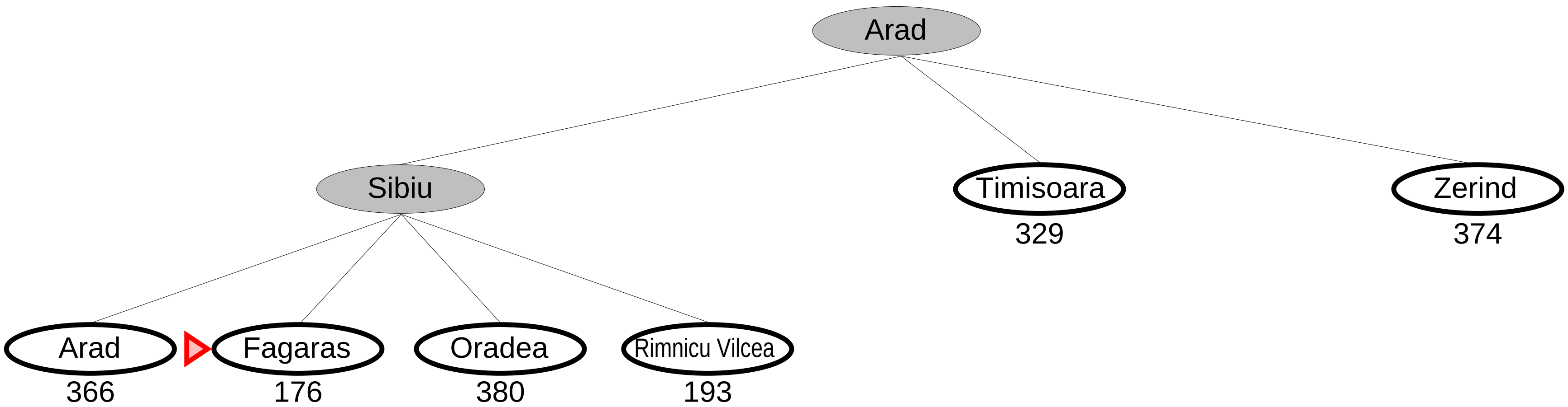
Greedy search expands the node that *appears* to be closest to goal

Greedy search example

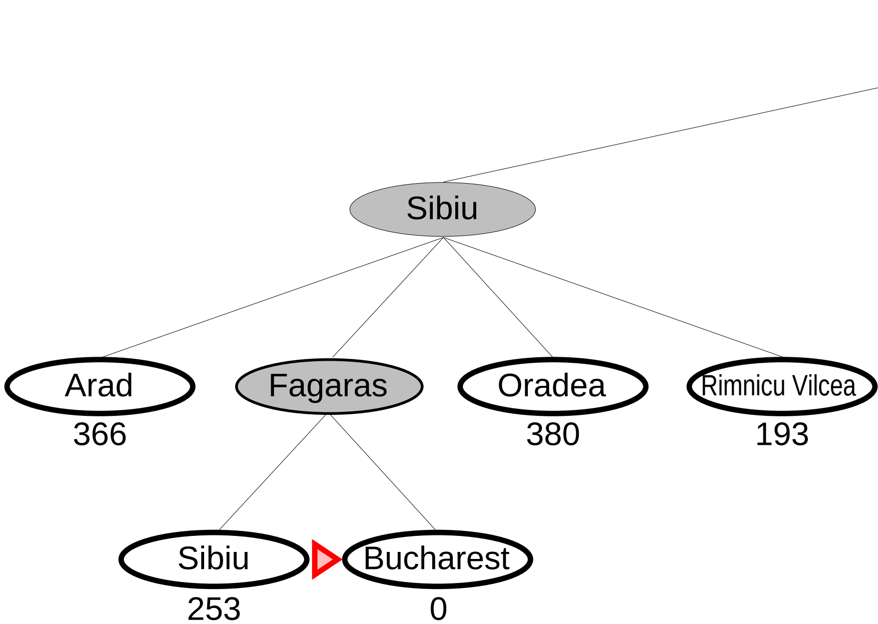
Greedy search example



Greedy search example



Greedy search example



Properties of greedy search

Complete??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g., with Oradea as goal,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,
Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$
Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal?? No

A^* search

Idea: avoid expanding paths that are already expensive

Evaluation function $f(n) = g(n) + h(n)$

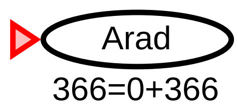
$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

A^* search uses an *admissible* heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the *true* cost from n.
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)

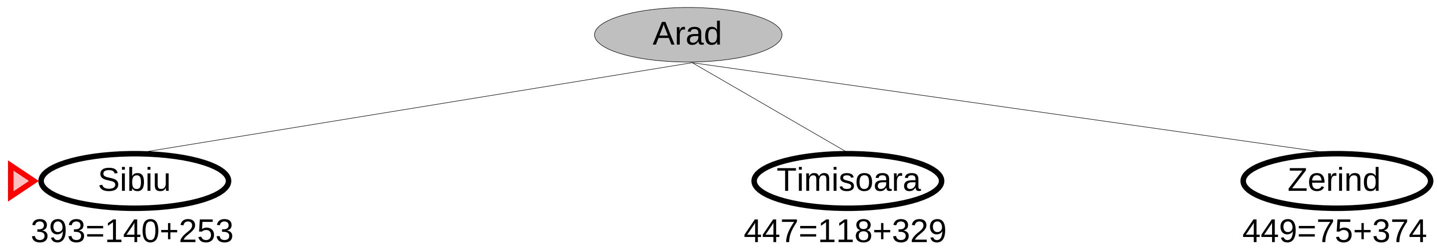
E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

Theorem: A^* search is optimal

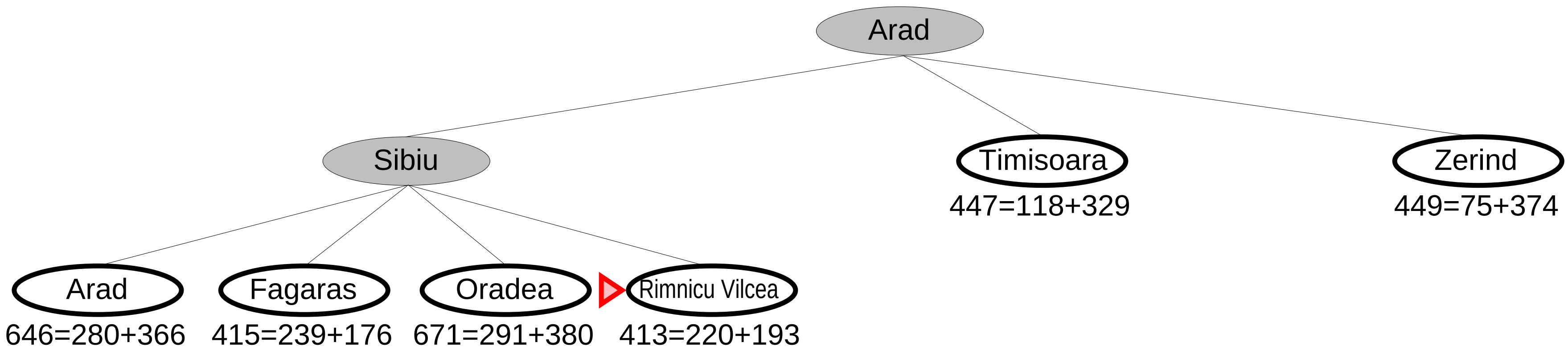
A* search example



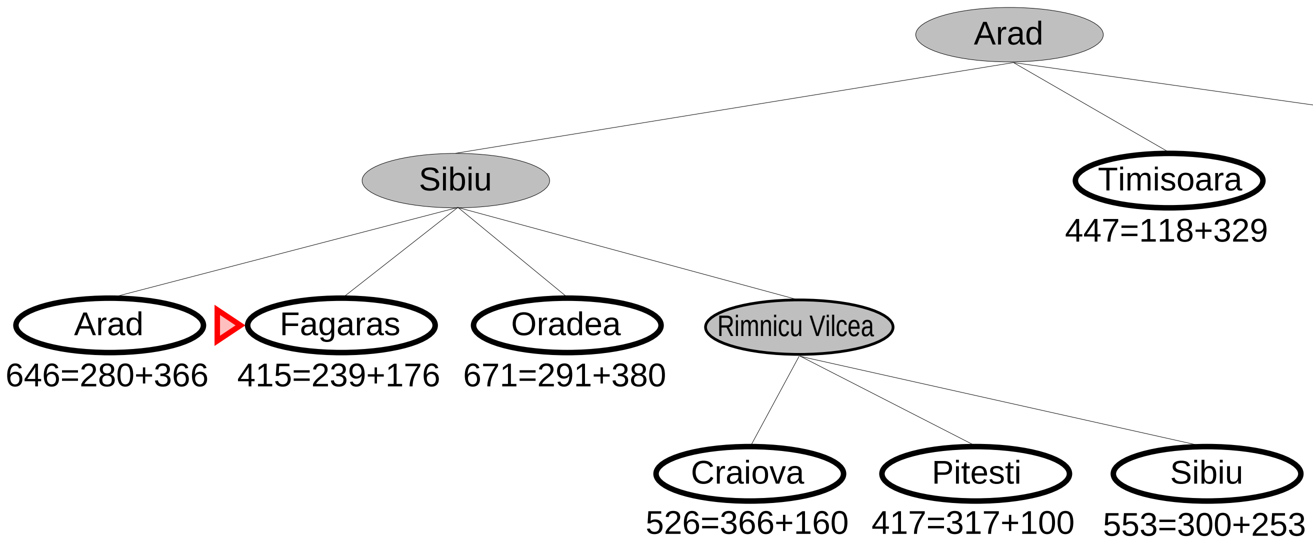
A* search example



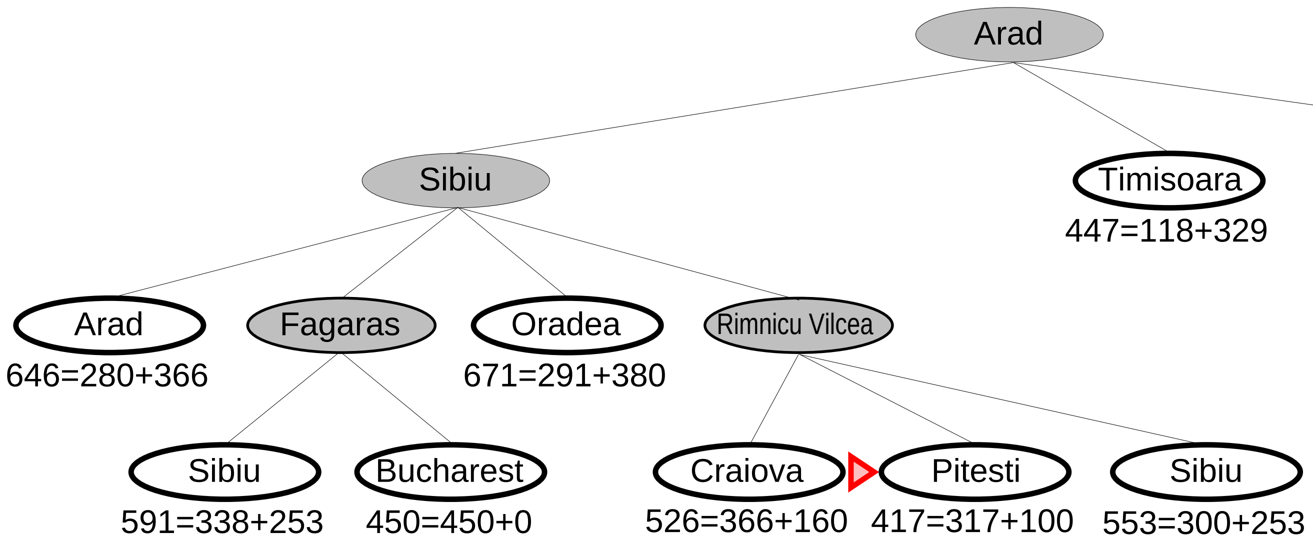
A* search example



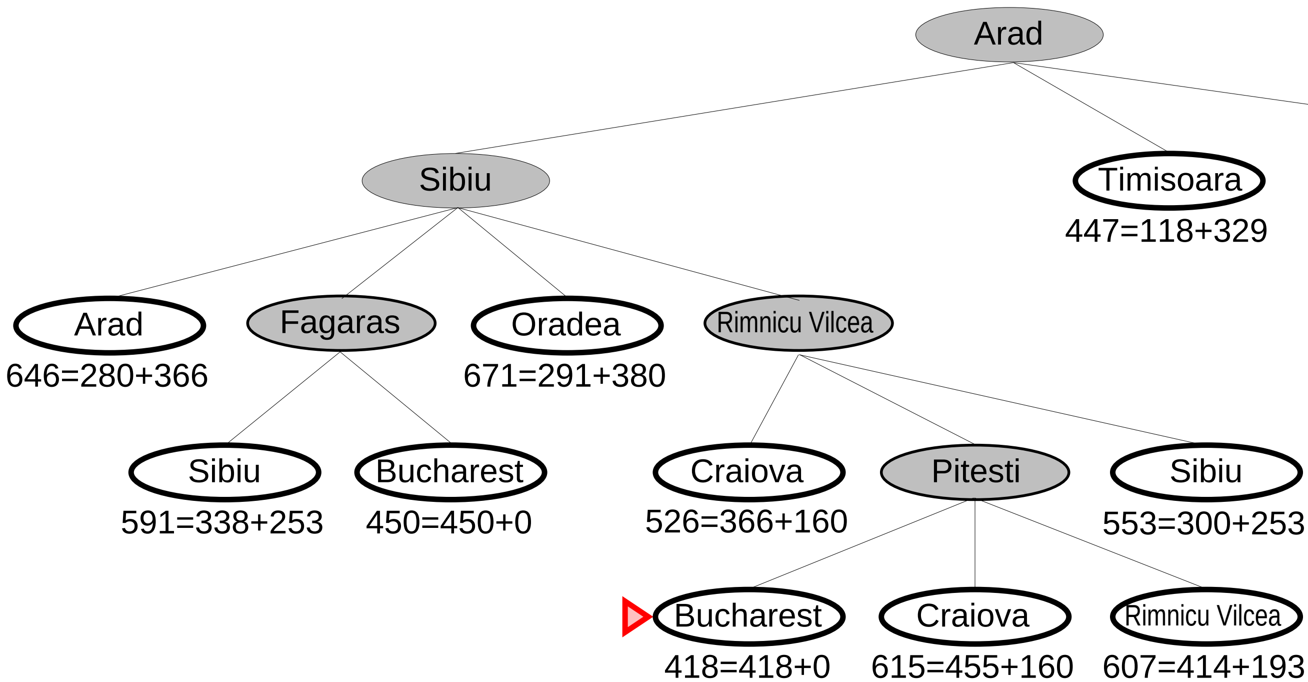
A* search example



A* search example

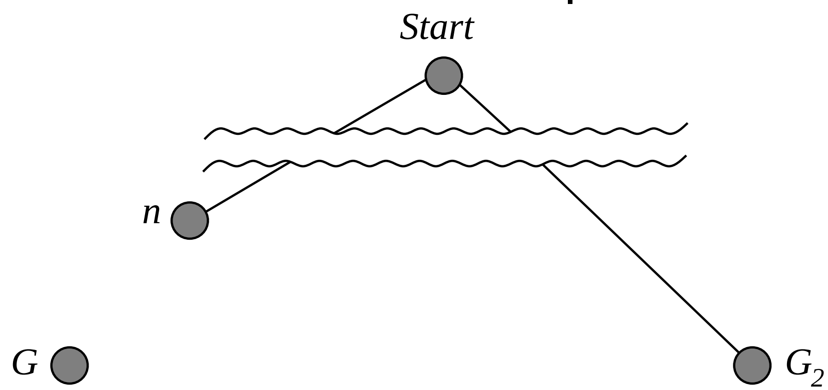


A* search example



Optimality of A* (standard proof)

Suppose some suboptimal goal G_2 has been generated and is in the queue.
Let n be an unexpanded node on a shortest path to an optimal goal G_1.



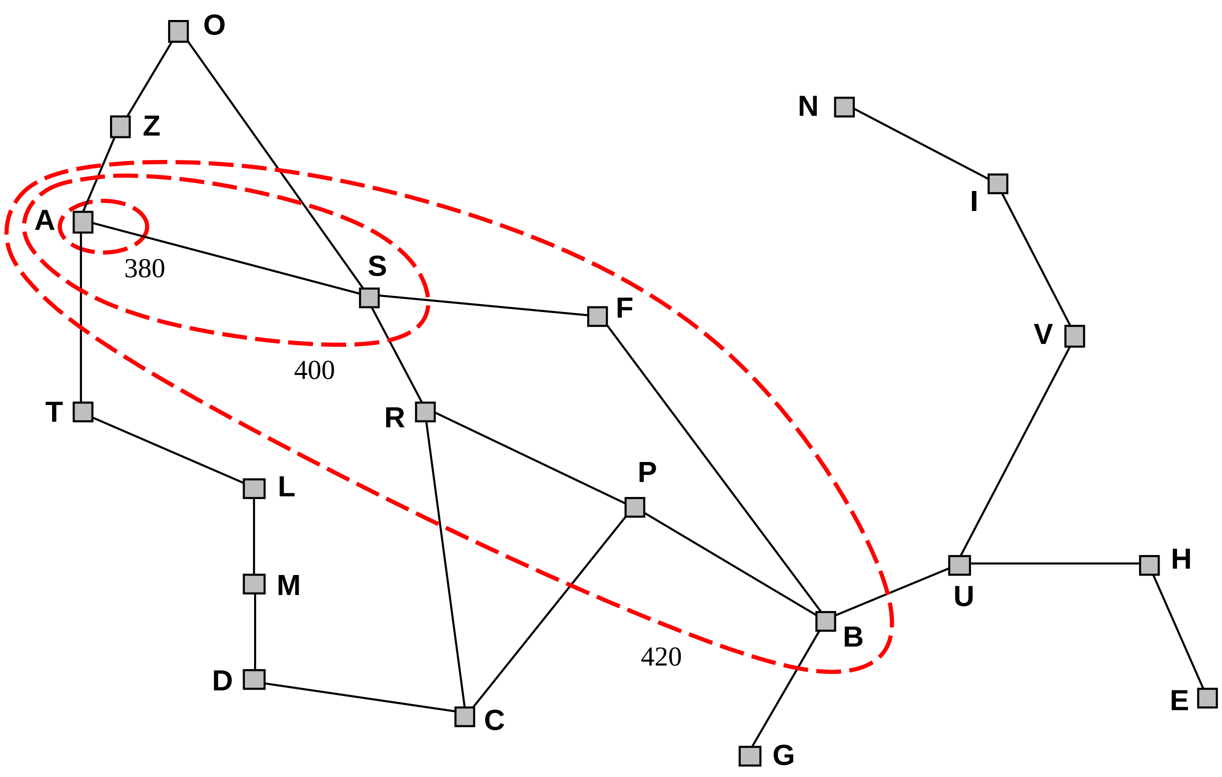
$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
&> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
&\geq f(n) && \text{since } h \text{ is admissible}
\end{aligned}
$$

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

Optimality of A* (more useful)

Lemma: A* expands nodes in order of increasing f value*

Gradually adds "f-contours" of nodes (cf. breadth-first adds layers)
Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



Properties of A*

Complete??

Properties of A^*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

Space??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

Space?? Keeps all nodes in memory

Optimal??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in $h \times$ length of soln.]

Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand f_{i+1} until f_i is finished

A* expands all nodes with $f(n) < C^*$
A* expands some nodes with $f(n) = C^*$
A* expands no nodes with $f(n) > C^*$

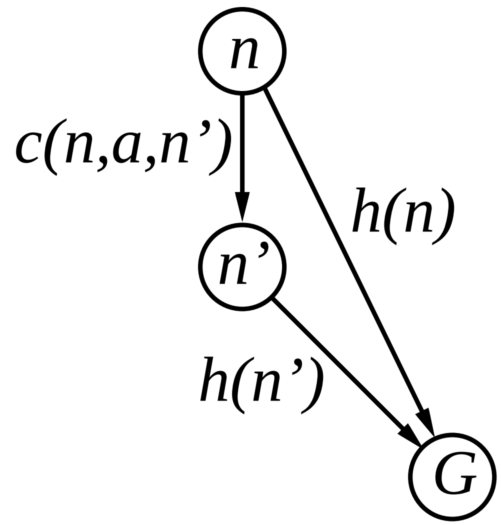
Proof of lemma: Consistency

A heuristic is *consistent* if

$$h(n) \leq c(n, a, n') + h(n')$$

If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

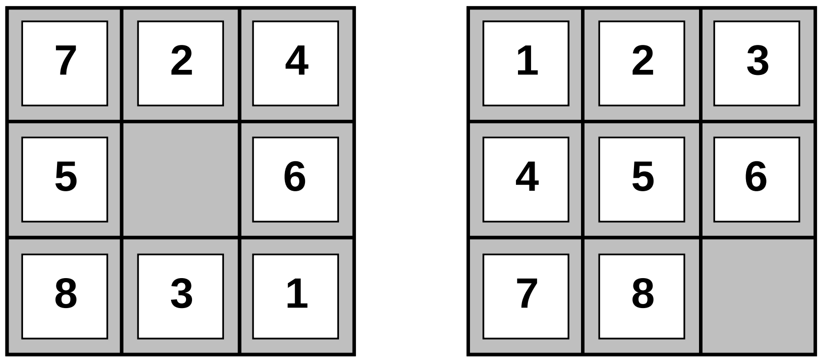


I.e., $f(n)$ is nondecreasing along any path.

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



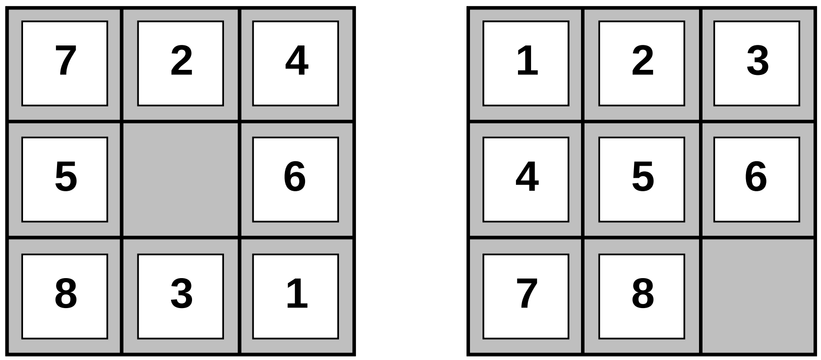
Start State **Goal State**

$h_1(S) =$??
$h_2(S) =$??

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



Start State **Goal State**

$\underline{h_1(S) =??}$ 7
$\underline{h_2(S) =??}$ 4+0+3+3+1+0+2+1 = 14

Dominance

If $h_2(n) \geq h_1(n)$ for all n (both admissible)
then h_2 *dominates* h_1 and is better for search

Typical search costs:

$d = 14$	IDS $=$ 3,473,941 nodes
	A$^*(h_1) =$ 539 nodes
	A$^*(h_2) =$ 113 nodes
$d = 24$	IDS $\approx$ 54,000,000,000 nodes
	A$^*(h_1) =$ 39,135 nodes
	A$^*(h_2) =$ 1,641 nodes

Hill-climbing (or gradient ascent/descent)

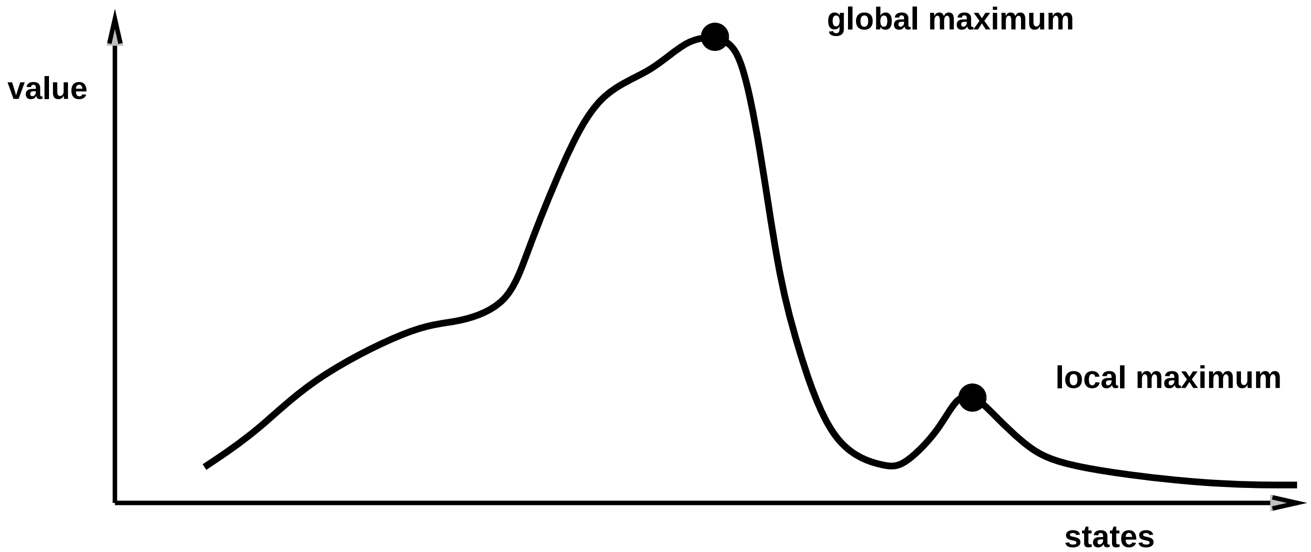
"Like climbing Everest in thick fog with amnesia"

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  inputs: *problem*, a problem
  local variables: *current*, a node
    neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  loop do
    neighbor ← a highest-valued successor of *current*
    if VALUE[neighbor] < VALUE[current] **then return** STATE[*current*]
    current ← *neighbor*
  end

Hill-climbing contd.

Problem: depending on initial state, can get stuck on local maxima



In continuous spaces, problems w/ choosing step size, slow convergence

Simulated annealing

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

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 prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow$ *schedule*[t]
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Properties of simulated annealing

At fixed "temperature" T, state occupation probability reaches Boltzman distribution

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

T decreased slowly enough $\Longrightarrow$ 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.

Board Games & Search

Move generation
Static Evaluation
Min Max
Alpha Beta
Practical matters

1949 Shannon paper
1951 Turing paper
1958 Bernstein program
55-60 Simon-Newell program
(α-β McCarthy?)
61 Soviet program
66 – 67 MacHack 6 (MIT AI)
70's NW Chess 4.5
80's Cray Blitz
90's Belle, Hitech, Deep Thought, Deep Blue

Types of games

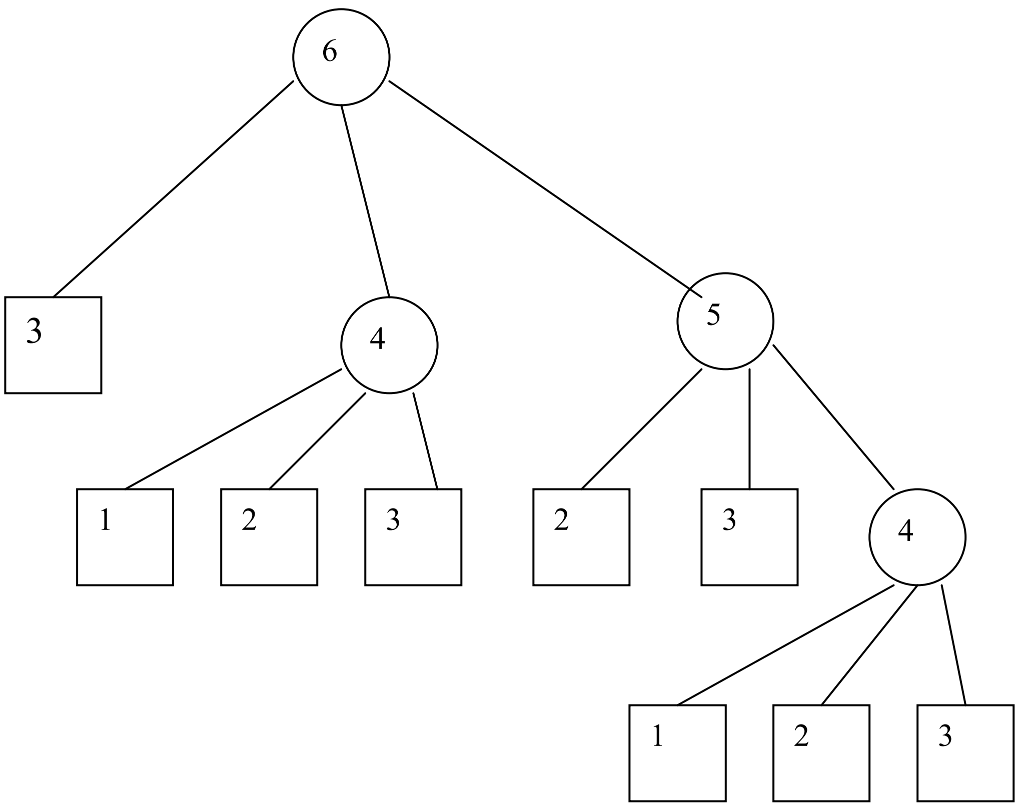
	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war

Game Tree Search

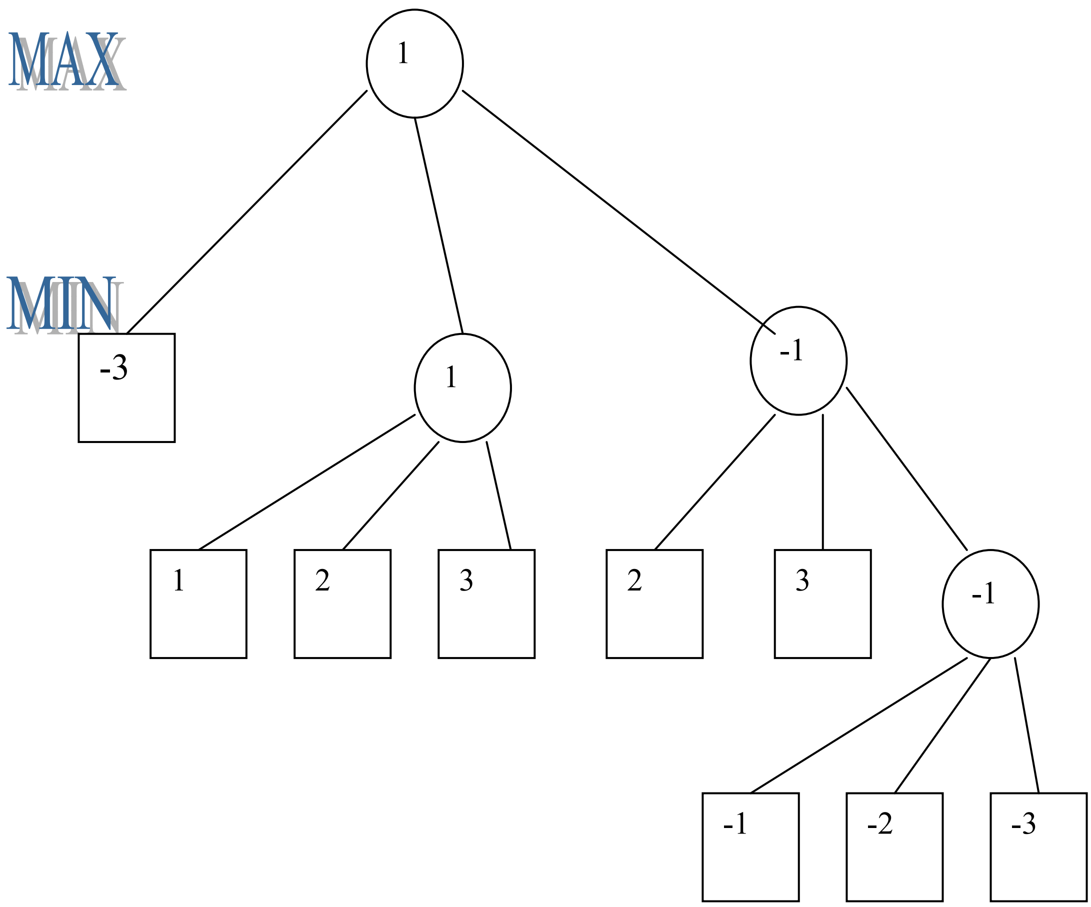
- Initial state: initial board position and player
- Operators: one for each legal move
- Goal states: winning board positions
- Scoring function: assigns numeric value to states
- Game tree: encodes all possible games

- We are not looking for a path, only the next move to make (that hopefully leads to a winning position)
- Our best move depends on what the other player does

A modified version of the game of "nim" :
Assume a pile that contains n chips in the beginning.
The first player can have three choices : take 1, 2 or 3 chips.
The second player can also have three choices : take 1, 2 or 3 chips.
The **winner** is the player who empties the pile first.
The amount of payoff is the number of chips the winner takes in his last turn.



Numbers shown indicate amount of **payoff**. Since one person's gain = another's person loss**, values representing the opponent scores are negated.**

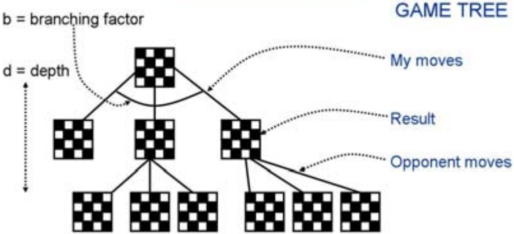


Best **strategy for MAX** player : take 2 chips and **always win** (if both **players play optimally**).

Other Games

- Backgammon
  - Involves randomness – dice rolls
  - Machine-learning based player was able to draw the world champion human player.
- Bridge
  - Involves hidden information – other players' cards – and communication during bidding.
  - Computer players play well but do not bid well
- Go
  - No new elements but huge branching factor
  - No good computer players exist

Move Generation

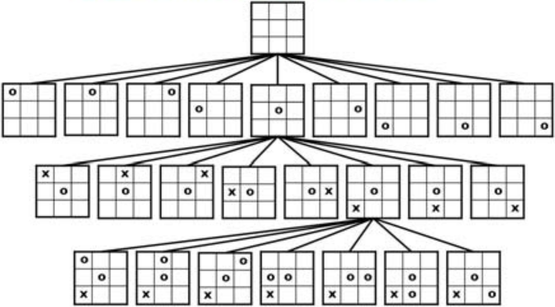


Chess

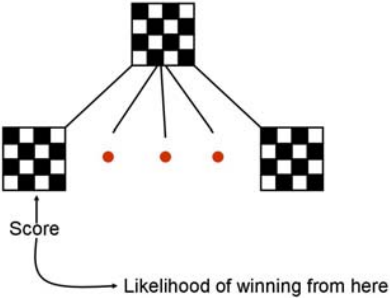
b = 36 | 36^{40} **is big!**

d > 40

Partial Game Tree for Tic-Tac-Toe



Scoring function



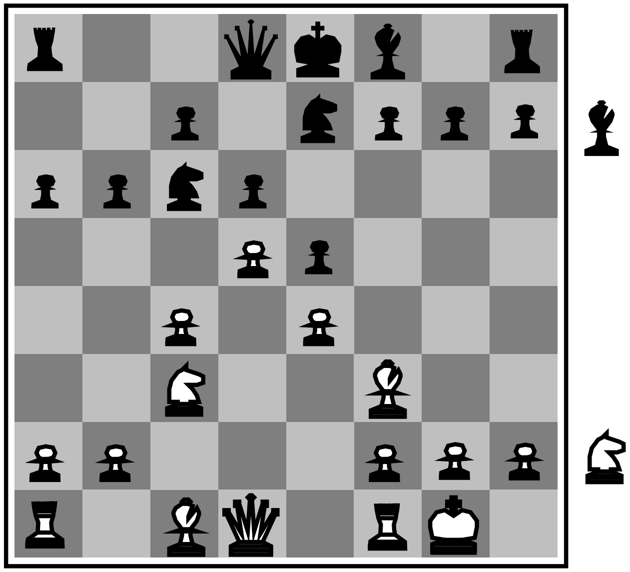
Static Evaluation

$$
\begin{aligned}
S = {} & c_1 \times \text{material} \\
+ {} & c_2 \times \text{pawn structure} \\
+ {} & c_3 \times \text{mobility} \\
+ {} & c_4 \times \text{king safety} \\
+ {} & c_5 \times \text{center control} \\
+ {} & \ldots
\end{aligned}
$$

P	1
K	3
B	3.5
R	5
Q	9

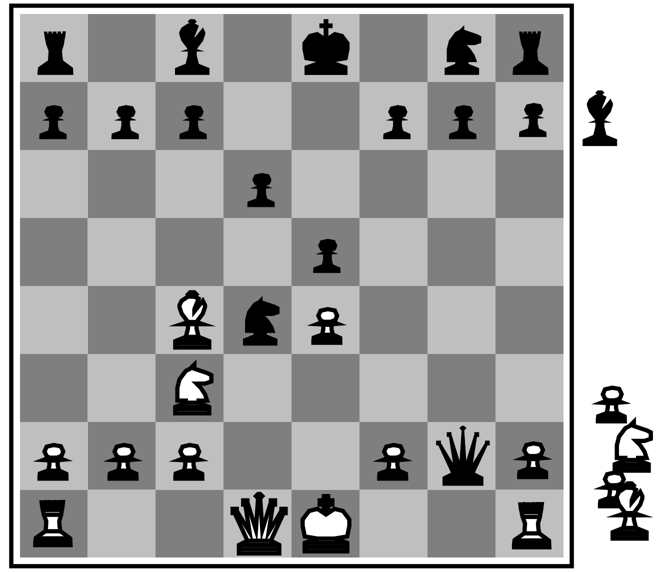
Too weak to predict ultimate success

Evaluation functions



Black to move

White slightly better



White to move

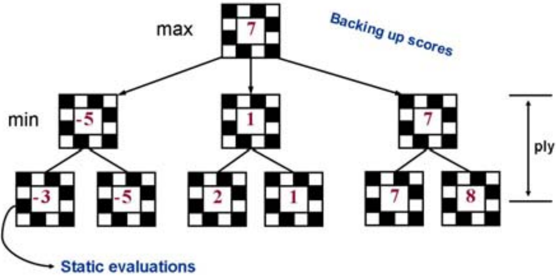
Black winning

For chess, typically *linear* weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

Limited look ahead + scoring



MIN-MAX

Min-Max

// initial call is MAX-VALUE(state,MAX-DEPTH)

```
function MAX-VALUE (state, depth)
   if (depth == 0) then return EVAL (state)
   v = -∞
   for each s in SUCCESSORS (state) do
      v = MAX (v, MIN-VALUE (s, depth-1))
   end
   return v

function MIN-VALUE (state, depth)
   if (depth == 0) then return EVAL (state)
   v = ∞
   for each s in SUCCESSORS (state) do
      v = MIN (v, MAX-VALUE (s, depth-1))
   end
   return v
```

Properties of minimax

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

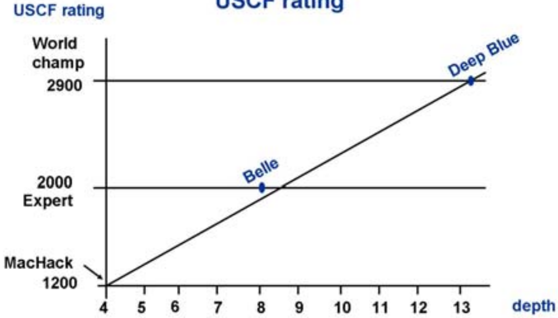
Optimal?? Yes, against an optimal opponent. Otherwise??

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

USCF rating

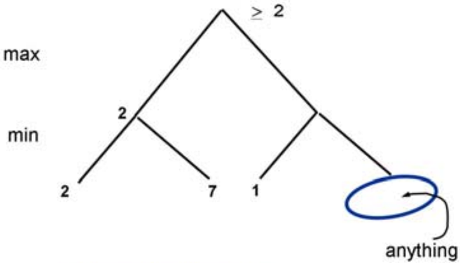

USCF rating
World champ
2900
2000
Expert
MacHack
1200
Deep Blue
Belle
4
5
6
7
8
9
10
11
12
13
depth

Deep Blue

32 SP2 processors
  each with 8 dedicated chess processors
= 256 CP

50 – 100 billion moves in 3 min
  13-30 ply search.

α - β



α is lower bound on score

β is upper bound on score

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?

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

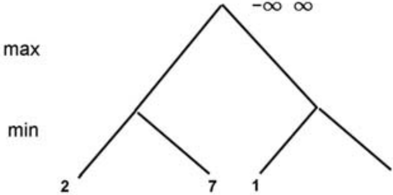
4-ply lookahead is a hopeless chess player!

4-ply $\approx$ human novice
8-ply $\approx$ typical PC, human master
12-ply $\approx$ Deep Blue, Kasparov

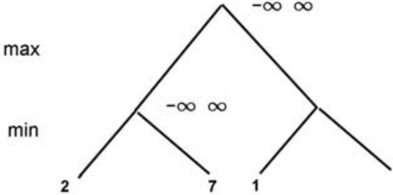
α - β

```
// α = best score for MAX,  β = best score for MIN
// initial call is MAX-VALUE(state,-∞,∞,MAX-DEPTH)

function MAX-VALUE (state, α, β, depth)
    if (depth == 0) then return EVAL (state)
    for each s in SUCCESSORS (state) do
        α = MAX (α, MIN-VALUE (s, α, β,depth-1))
        if α ≥ β then return α // cutoff
    end
    return α

function MIN-VALUE (state, α, β, depth)
    if (depth == 0) then return EVAL (state)
    for each s in SUCCESSORS (state) do
        β = MIN (β, MAX-VALUE (s, α, β,depth-1))
        if β ≤ α then return β // cutoff
    end
    return β
```

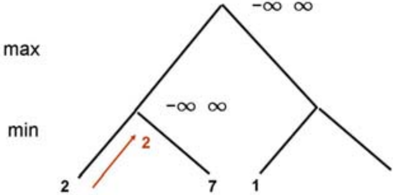
α - β

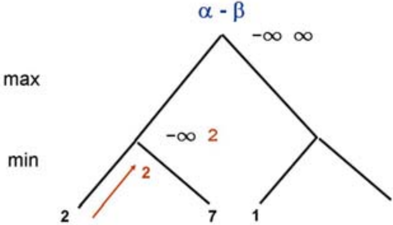


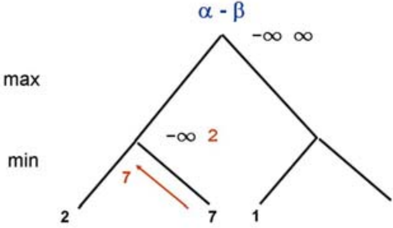
α - β



α - β

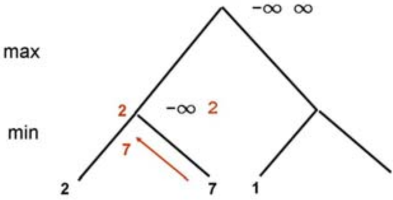


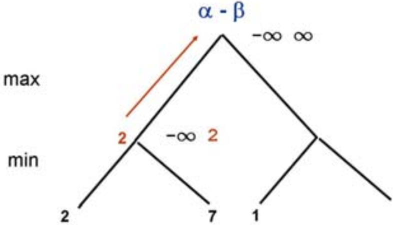

α - β
−∞ ∞
max
−∞ 2
min
2
2
7
1

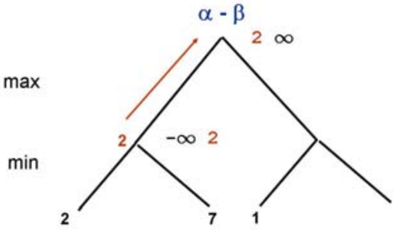


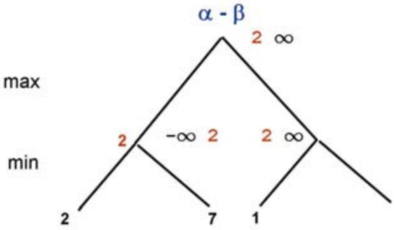
α - β
−∞ ∞
max
−∞ 2
min
7
2
7
1

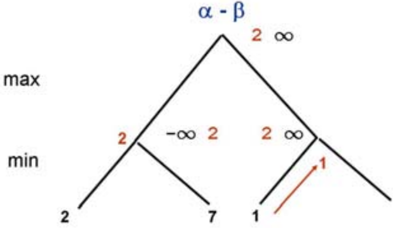
α - β



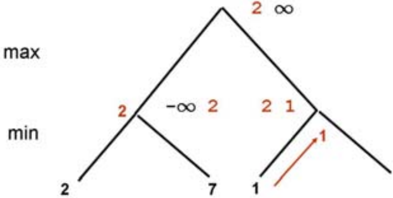

α - β
−∞ ∞
max
2
−∞ 2
min
2
7
1

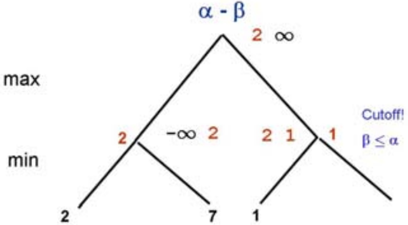

α - β
2 ∞
max
2
-∞ 2
min
2
7
1


α - β
2 ∞
max
2
-∞ 2
2 ∞
min
2
7
1

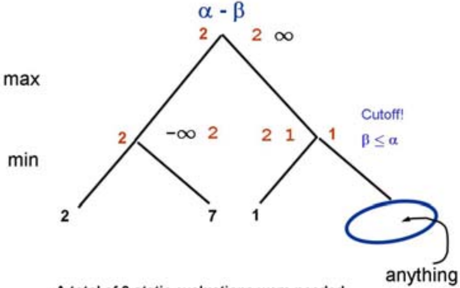

α - β
2 ∞
max
2
-∞ 2
2 ∞
min
1
2
7
1

α - β



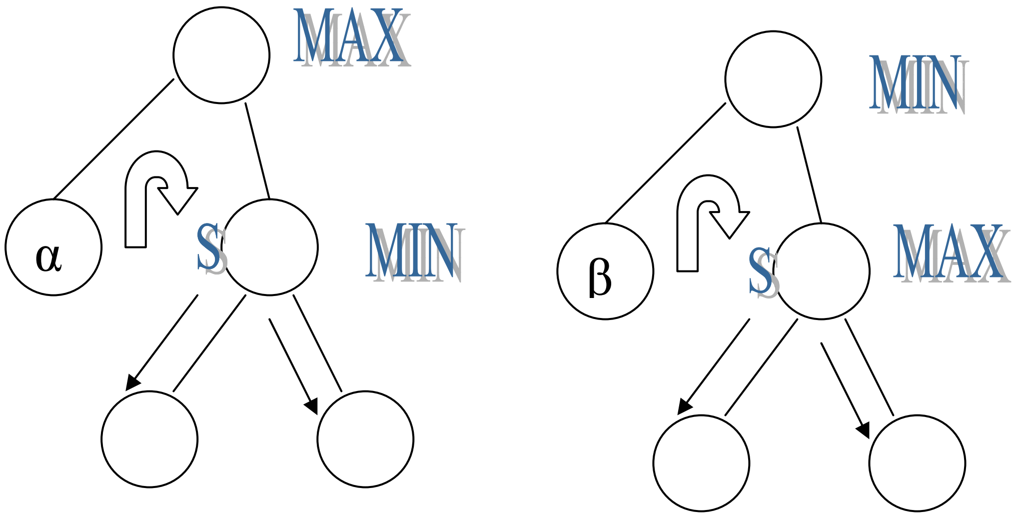


α - β
2 ∞
max
Cutoff!
β ≤ α
2
−∞ 2
2 1
1
min
2
7
1



A total of 3 static evaluations were needed to obtain the value for the tree.

<u>Idea of α – β Pruning</u> :-



α Cut-Off : If we know that node S has a value $\leq$ **α** then prune the tree with S as root.

α Can not Decrease

β Cut-Off : If we know that node S has a value $\geq$ **β** then prune the tree with S as root.

β Can not Increase

Whenever the **α** cut-off exceeds the **β** cut-off we use the **α** cut-off if the node is **MAX**

& use the **β** cut-off if the node is **MIN**

α - β

1. Guaranteed same value as Max-Min
2. In a perfectly ordered tree, expected work is $0(b^{d/2})$, vs $0\ (b^d)$ for Max-Min, so can search twice as deep with the same effort!
3. With good move ordering, the actual running time is close to the optimistic estimate.

Properties of α–β

Pruning *does not* affect final result

Good move ordering improves effectiveness of pruning

With "perfect ordering," time complexity $= O(b^{m/2})$
$\Rightarrow$ *doubles* depth of search
$\Rightarrow$ 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*)

α - β (NegaMax form)

```
// α = best score for MAX,  β = best score for MIN
// initial call is Alpha-Beta(state,-∞,∞,MAX-DEPTH)

function Alpha-Beta (state, α, β, depth)
    if (depth == 0) then return EVAL (state)
    for each s in SUCCESSORS (state) do
        α = MAX(α, -Alpha-Beta (s, -β, -α, depth-1))
        if α ≥ β then return α // cutoff
    end
    return α
```

Game Program

	Time
1. Move generator (ordered moves)	50%
2. Static evaluation	40%
3. Search control	10%

openings
end games
> databases

[all in place by late 60's.]

Move Generator

1. Legal moves
2. Ordered by
    1. Most valuable victim
    2. Least valuable agressor
3. Killer heuristic

Static Evaluation

Initially - Very Complex

70's - Very simple (material)

now -

- Deep searchers: moderately complex (hardware)
- PC programs: elaborate, hand tuned

Practical matters

Variable branching

- Iterative deepening
  - order best move from last search first
  - use previous backed up value to initialize $[\alpha, \beta]$
  - keep track of repeated positions (transposition tables)

Horizon effect

- quiescence
- Pushing the inevitable over search horizon

Parallelization

OBSERVATIONS

- Computers excel in well-defined activities where rules are clear
  - chess
  - mathematics
- Success comes after a long period of gradual refinement

For more detail on building game programs visit:

http://www1.ics.uci.edu/~eppstein/180a/w99.html