2 Problem Solving and Search

Let's have Holiday in Romania ☺

- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow from Bucharest at 1:00.
- Let's configure this to be an AI problem.

What is the Problem?

- Accomplish a *goal*
  - Reach Bucharest by 1:00
- So this is a goal-based problem

Is there more?

- Do it well… according to a *performance measure*
  - Minimize distance traveled

Examples of a non-goal-based problem?

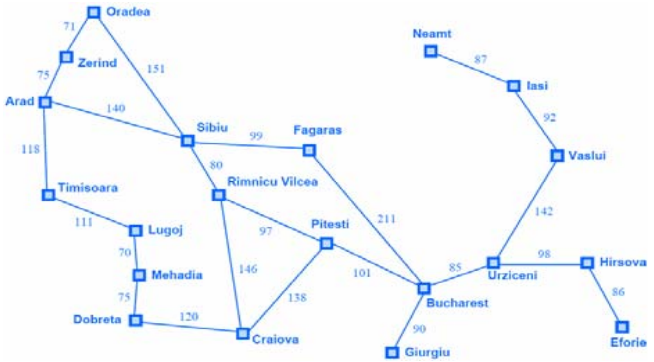
- Live longer and prosper
- Maximize the happiness of your trip to Romania
- Don't get hurt too much

What Qualifies as a Solution?

- You can/cannot reach Bucharest by 1:00
- You can reach Bucharest in *x* hours
- The shortest path to Bucharest passes through these cities
- The sequence of cities in the shortest path from Arad to Bucharest is _____
- The actions one takes to travel from Arad to Bucharest along the shortest path

What additional information does one need?

- A map



Problem Definition and Solution Process

- The first task when solving any problem is the *precise definition of the problem*
- *Well-defined Problems*
  - *A solution exists*
  - *It is unique*
  - *The solution does not change with initial state*
- Formal definition of a problem
  - **Initial state**
  - Description of possible and available **Actions**
  - **Goal Test** which determines whether a given state is the goal
  - **Path Cost** assigns a numeric cost to each path

A simplified Road Map

- A solution to problem is a path from the initial state to goal state
- Solution quality is measured by the path cost

Abstraction

- Proposed formulation of the problem seems reasonable
- But in actual cross-country trip, the states of the world includes so many things:
  - the traveling companions,
  - what is on the radio,
  - how far away it is to the next rest stop,
  - any traffic law enforcements,
  - the weather and road conditions, ...
- All these considerations are left out of our state description of the road because they are irrelevant to the problem of finding a route to Bucharest
- The process of removing detail from a representation is called **abstraction**

Abstraction Level

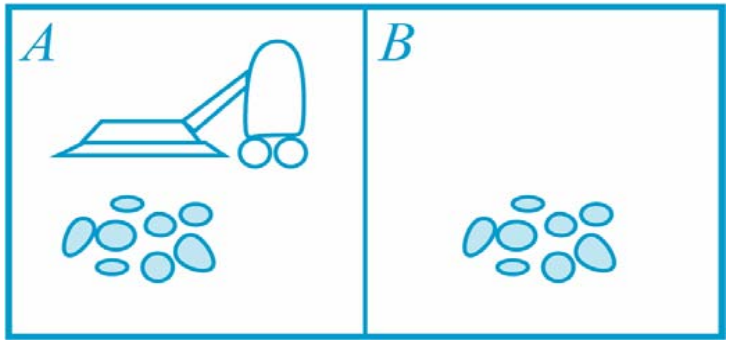
- Can we more precise about defining the appropriate level of abstraction?
- The abstraction is valid if we can expand any abstract solution into a solution in the more detailed world.
- The abstraction is useful if carrying out each of the actions in the solution is easier than the original problem
- The choice of a good abstraction involves removing as much detail as possible while retaining validity and ensuring that the abstract actions are easy to carry out.

Example Problems

- The problem-solving approach has been applied to a vast array of task environments
- We list some of the best known here
- Toy Problems
  - Vacuum world
  - 8-puzzle Problem
  - 8-queens Problem
- Real-world Problems
  - TSP
  - VLSI Layout
  - Robot Navigation

Vacuum World

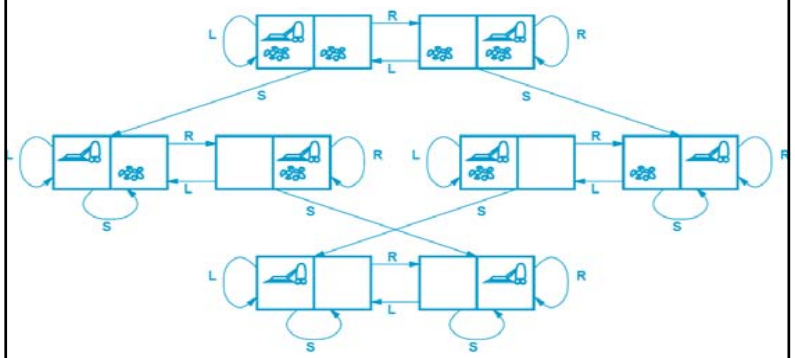
- A vacuum-cleaner world with just two locations



Vacuum World

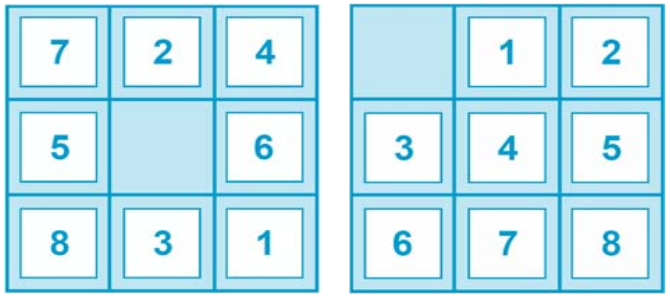
- **States**: The cleaner is in one of two locations, each of which might or might not contain dirt
  - Thus there are $2 \times 2^2 = 8$ possible world states
- **Initial State**: Any state can be designated as the initial state
- **Actions**: *Left*, *Right*, *Suck*
- **Goal Test**: All the squares are clean
- **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

The State Space for the Vacuum World



8-Puzzle

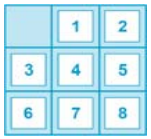
- The 8-puzzle consists of 3×3 board with eight numbered tiles and a blank space
- A tile adjacent to the blank space can slide into blank space



Initial State **Goal**

8-Puzzle

- **States**: location of each of the eight tiles and the location of the blank space
  - Thus there are 9!/2=181,440 reachable states
- **Initial State**: Any state can be designated as the initial state
- **Actions**: *Left*, *Right*, *Up*, *Down*
- **Goal Test**:



- **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

Abstraction Revisited

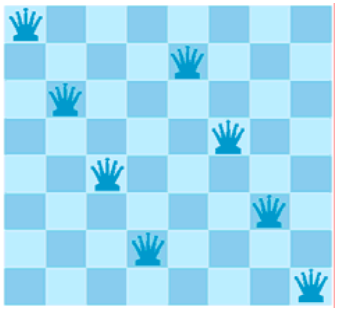
- What abstraction have we included here?
  - we ignore the intermediate locations where the block is sliding
  - we abstract away actions such as shaking the board when pieces get stuck, or extracting pieces with a knife and putting them back again
- we avoid all the details of physical manipulations

More on 8-Puzzle Problem

- The 8-puzzle belongs to the family of **sliding-block puzzle**, which are often used to test problems for new search algorithms in AI
- This general class is known to be NP-complete
  - One does not expect to find methods significantly better in the worst case than the search algorithms that we will study in the course
- The 8-puzzle (3×3 board): **181,440** states
- The 15-puzzle (4×4 board): **1,3 trillion** states
  - can be solved in milliseconds by the best search algorithm
- The 24-puzzle (5×5 board): ~10^{25} states
  - quite difficult to solve with current machines

8-Queens Problem

- The goal of 8-Queens problem is to place eight queens on a chessboard such that no queen attacks any other



8-Queens Problem

- **States**: Any arrangement of 0 to 8 queens on the board
  - Thus there are $64 \times 63 \ldots 57 \approx 3 \times 10^{14}$ possible sequences
- **Initial State**: No queens on the board
- **Actions**: *Add a queen to any empty square*
- **Goal Test**: 8 queens are on the board, none attacked

Another Representation

- **States**: Arrangement of *n* queens, one per column in the leftmost *n* columns, with no queen attacking another
  - Thus there are just 2057 possible sequences
- **Initial State**: No queens on the board
- **Actions**: *Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen*
- **Goal Test**: 8 queens are on the board, none attacked

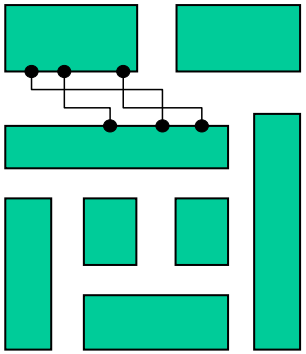
Real-world Problems

- Traveling Salesperson Problem (TSP)
  - a touring problem in which each city must be visited exactly once
  - the aim is to find the shortest tour
  - the problem is known to be NP-Hard
- VLSI Layout Problem
  - requires positioning millions of components an connections on a chip to minimize area, minimizing circuit delays, minimize stray capacitances, maximizing manufacturing yield
  - comes after logical design
  - split into two parts: **cell layout** and **channel routing**

Real-world Problems (con't)

- VLSI Layout Problem
  - **Cell Layout Problem**
    - Each cell has a fixed footprint (size and shape)
    - certain number of connection to other cells
    - the aim is to place the cells on the chip so that they do not overlap and there is room for the connecting wires to be placed between cells
  - **Channel Routing Problem**
    - finding a specific route for each wire through the gaps between the cells

Cell Layout Problem

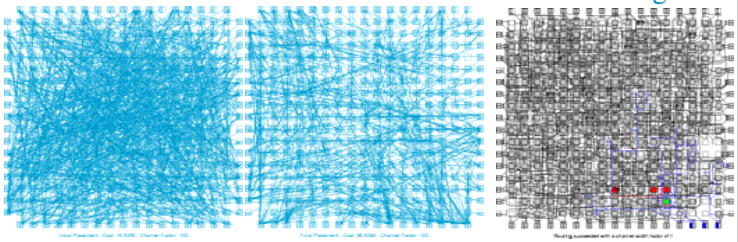


Channel Routing

Random Initial Placement

Final Placement

After Detailed Routing



Real-world Problems (con't)

- Robot Navigation
  - generalization of the route-finding problem described earlier
  - rather than a discrete set of routes, a robot can move in a continuous space with an infinite set of possible actions and states

The 25-pound, six-wheeled robotic explorer

Searching for Solutions

- We have formulated some problems, we now need to solve them ☹
- This is done by a search through the state space
- Here we deal with search techniques that use an explicit **search tree**
- Search tree is generated by the initial state and actions that together define the state space

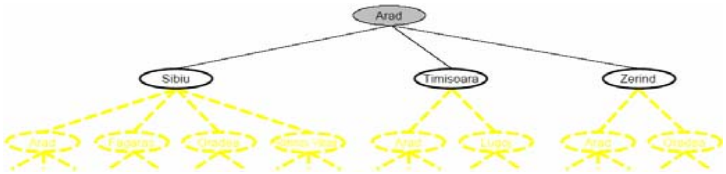
Search Node and Expanding Tree

- The root of the search tree is a search node corresponding to the initial state
- The first step is to test whether this is a goal state
- if this is not a goal state, we need to consider some other states. This is done by expanding the current state

Initial State

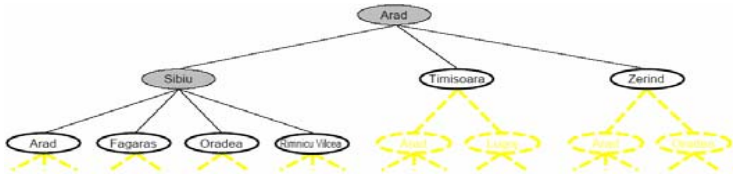
Search Node and Expending Tree (Con't)

- After expanding Arad



Search Node and Expending Tree (Con't)

- After expanding Sibiu



- We continue choosing, testing, and expanding until either a solution is found or there are no more states to be expanded.
- The choice of which state to expand is determined by the **search strategy**.

State Space *vs* Search Tree

- It is important to distinguish between the state space and the search tree
- For the route finding problem, there are only 20 states in the state space, one for each city
- But there are an infinite number of paths in this state space
- Some of these paths are repeated, so it is important to avoid from such repetitions

Measuring Problem-Solving Performance

- The output of the search algorithm is either failure or a solution
- We evaluate the searching performance in four ways:
  - **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
  - **Optimality**: Does the strategy find the optimal solution?
  - **Time complexity**: How long does it take to find a solution?
  - **Space complexity**: How much memory is needed to perform the search?

Branching Factor and Search Cost

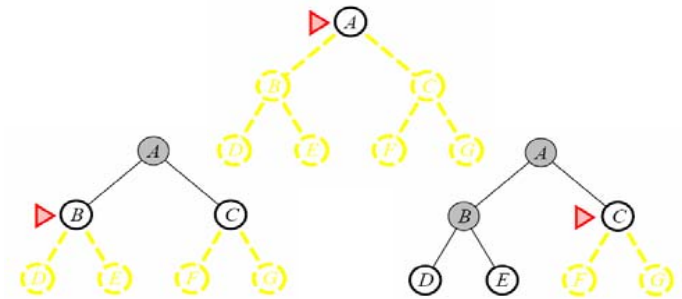
- Time and Space Complexity are related with respect to some measure of the Problem Difficulty
- Complexity is expressed in terms of three quantities:
  - b: the branching factor
  - d: depth of the shallowest goal node
  - m: maximum length of any path in the state space
- Time is measured in terms of nodes generated during the search
- Space complexity is measured in terms of maximum number of nodes stored in the memory
- Search cost typically depends on the time complexity

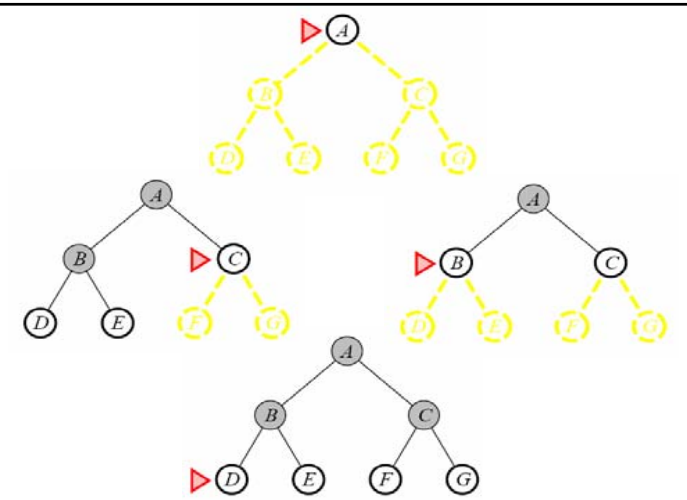
Uninformed Search Strategies

- a.k.a. Blind Search
- We will cover five blind search strategies
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening depth-first search

Breadth-first Search

- Simple strategy: the root node is expanded first, then all the successors of the root are expanded next, then their successors, and so on.





Analysis of Breadth-first Search

- It is complete
- In the worst case, the total number of nodes generated is

$$b + b^2 + b^3 + \cdots + b^d + \left(b^{d+1} - b\right) = O\left(b^{d+1}\right)$$

Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 MB
4	11100	11 seconds	106 MB
6	10^7	19 minutes	10 GB
8	10^9	31 hours	1 TB
10	10^{11}	129 days	101 TB
12	10^{13}	35 years	10 PetaBytes
14	10^{15}	3523 years	1 ExaByte

Analysis of Breadth-first Search (Con't)

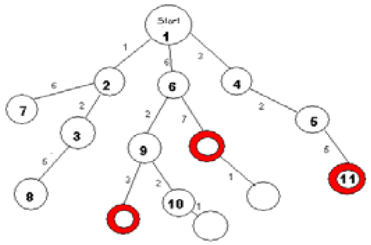
- There are two conclusions from the previous table
  - the memory requirements are a bigger problem for breadth-first search than is the execution time.
  - the time requirements are still a major factor

Uniform-cost Search

- Breadth-first search is optimal when all step costs are equal

  (it always expands the shallowest unexpanded node)
- By a simple extension, instead of expanding the shallowest node, uniform-cost search expands the node *n* with the lowest path cost

An Example



Uniform-cost search

- Complete? *If step cost ≥ ε*
- Time Complexity
  - *If all costs are equal =* $O(b^d)$
- Space Complexity

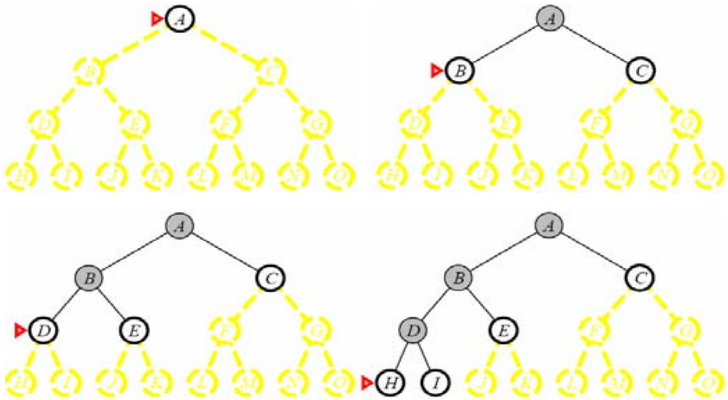
  $O\left(b^{\lceil C^*/\varepsilon \rceil}\right)$
- *Optimal?*

  Yes–nodes expanded in increasing order

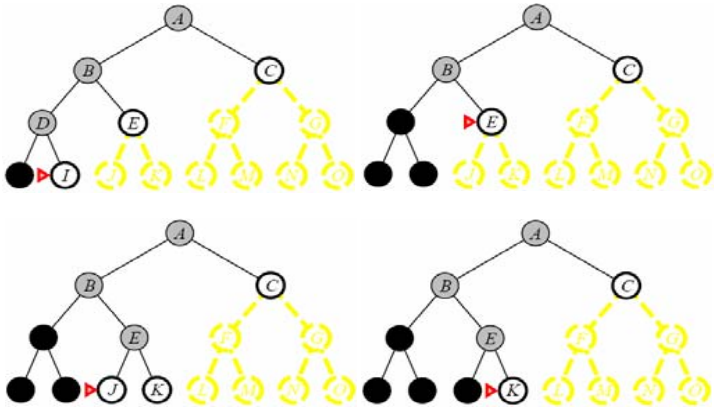
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *Fringe* = LIFO queue, i.e., a stack
  - Execute first few expansions of Arad to Bucharest using Depth-first search

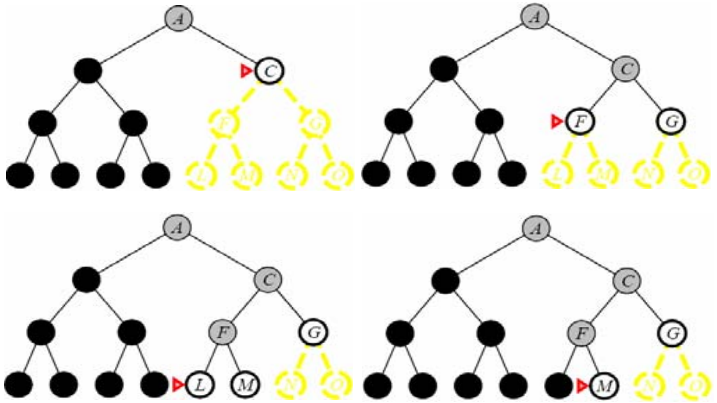
Depth-first search



Depth-first search



Depth-first search



Depth-first search

- Complete
  - No: fails in infinite-depth spaces, spaces with loops.
  - Can be modified to avoid repeated states along path ➔ complete in finite spaces
- Time Complexity
  - $O(b^m)$: terrible if m is much larger than d, but if solutions are dense, may be much faster than breadth-first
- Space Complexity
  - *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
- Depth-firt search is a special case of depth-limited search with ***l=∞***
- Solves the infinite-path problem
- However it also introduces an additional source of incompleteness if we choose ***l<d***
- This is likely when ***d*** is unknown
- Sometimes depth limit can be based on knowledge of the problem: ***diameter*** of the state space

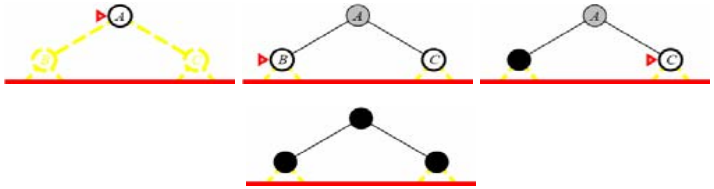
Iterative Deepening Search

- often used in combination with depth-first, that finds the best depth limit
- this is done by gradually increasing the limit —first 0, then 2, and so on— until a goal is found.

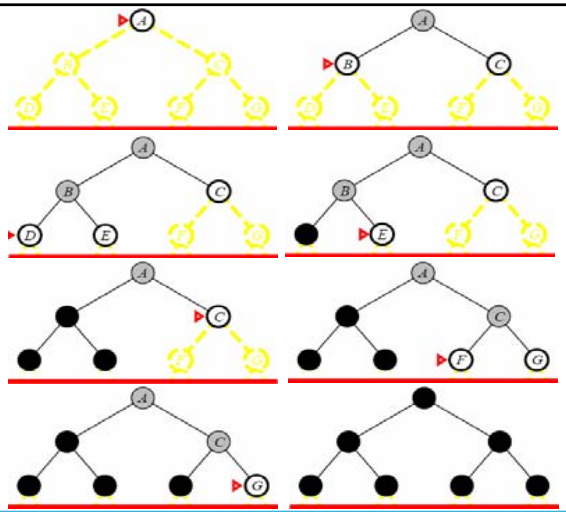
4 Iterations of Iterative Deepening Search on BT

Limit=0

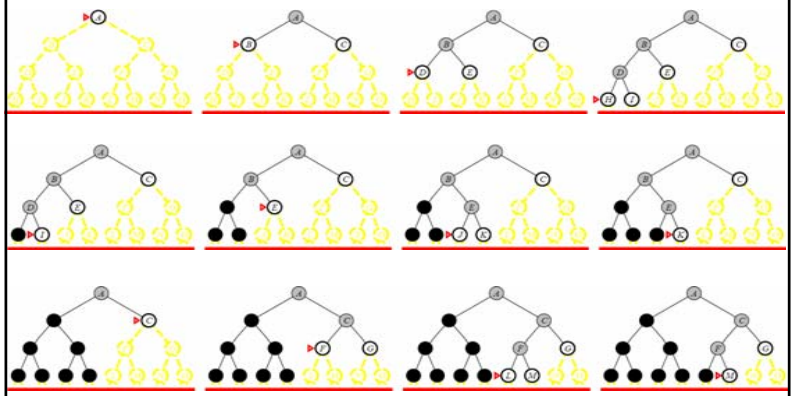
Limit=1



Limit=2



Limit=3

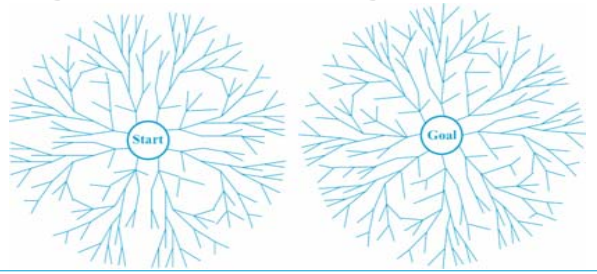


Properties of iterative deepening

- Complete
  - Yes
- Time Complexity
  - $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots b^d = O(b^d)$
- Space Complexity
  - $O(bd)$
- Optimal?
  - If step cost = 1
  - Can be modified to explore uniform-cost tree

Bidirectional Search

- Run two simultaneous searches
  - one **forward** from the initial
  - the other **backward** from the goal
  - **stops** when the two searches meet in the middle
- Two $b^{d/2}$ searches instead of one b^d



Bidirectional Search

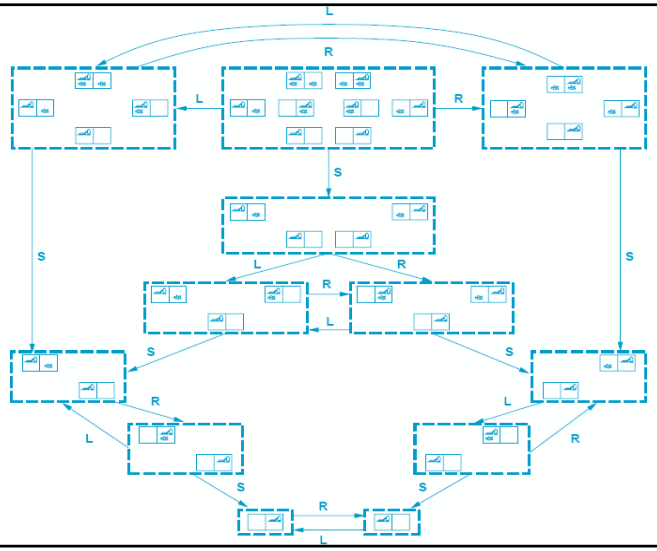
- Implementation
  - Each search checks nodes before expansion to see if they are on the fringe of the other search
  - Example: Bidirectional BFS search with **d=6** & **b=10**
    - Worst case: both search trees must expand all but one element of the third level of the tree
      - $2\times(10^1 + 10^2 + 10^3 + 10^4 - 10)$ node expansions
      - *Versus* $1 * (10^1 + 10^2 + \ldots + 10^7)$ expansions

Avoiding Repeated States

- So far, we ignored one of the most important complications to the search process: expanding already expanded states
- Repeated states are unavoidable for some problems
  - e.g. route finding, 8-puzzle
  - search trees are infinite for these problems

Bidirectional Search

- Implementation
  - Checking fringe of other tree
    - At least one search tree must be kept in memory
    - Checking can be done in constant time (hash table)
  - Searching back from goal
    - Must be able to compute predecessors to node n: *Pred (n)*
    - Easy with 15-puzzle, but how about chess?



Sensorless Problems

- Suppose that the vacuum agent knows all the effects of its actions, but has no sensors
- Then it does not know its initial state
- It sometimes happens when the world is not fully observable
- In this case the agent must reason about the states that it might get to, rather than single state
- The set of states is called **Blief State**
- To solve sensorless problems, we search in the space of belief states rather than physical states

Informed Search

- INFORMED?
- Uses problem-specific knowledge beyond the definition of the problem itself
  - selecting best lane in traffic
  - playing a sport
    - what's the heuristic (or evaluation function)?

Best-first Search

- BFS/DFS/UCS differ in how they select a node to pull off the fringe
  - We want to pull the node that's on the optimal path off the fringe
    - But if we know the "best" node to explore, we don't have to search!
- Use an evaluation function to select node to expand
  - f(n) = evaluation function = expected "cost" for a path from root to goal that goes through node *n*
  - Select the node *n* that minimizes *f(n)*
  - How do we build *f(n)*?

Evaluation function: *f(n)*

- Combine two costs
  - *f(n)* = *g(n)* + *h(n)*
    - *g(n)* = cost to get to *n* from *start*
    - *h(n)* = cost to get to from *n* to *goal*

Heuristics

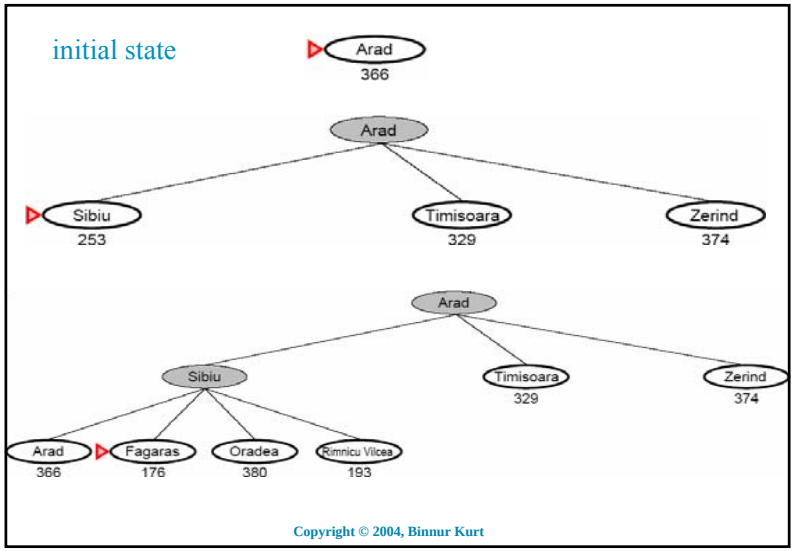
- A function, h(n), that estimates cost of cheapest path from node ***n*** to the goal
  - ***h(n) = 0*** if ***n*** == goal node

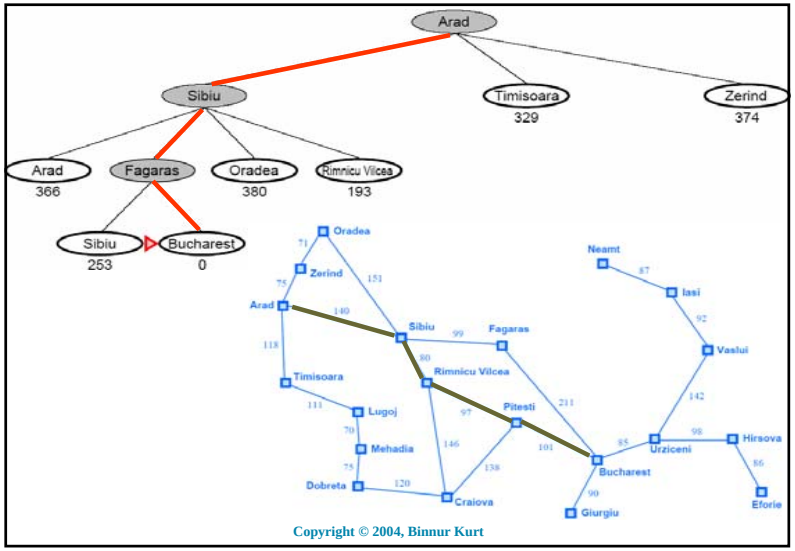
Greedy Best-first Search

- Trust your heuristic
  - evaluate node that minimizes *h(n)*
  - *f(n)* = *h(n)*
- Example: getting from A to B
  - Explore nodes with shortest straight distance to B
  - Shortcomings of heuristic?
    - Greedy can be bad…
    - Climbing K2 accomplishes a goal, but the cost of getting there is prohibitive!

h(*n*): Straight Distance

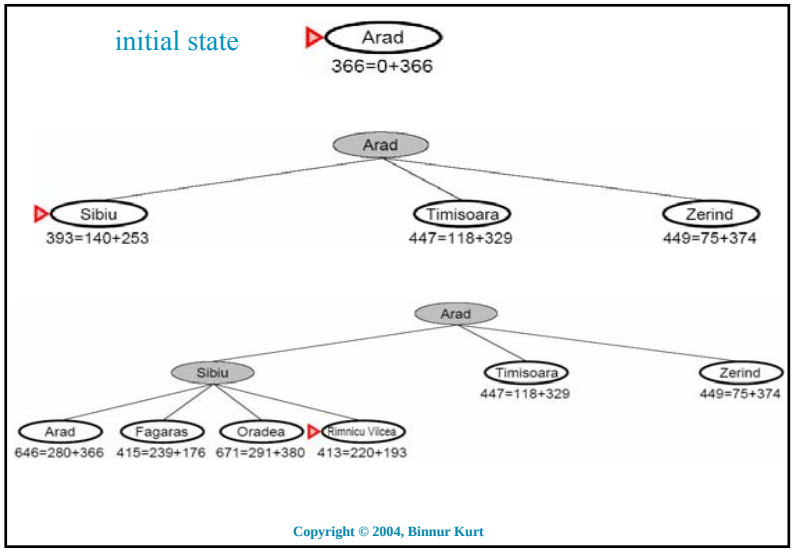
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

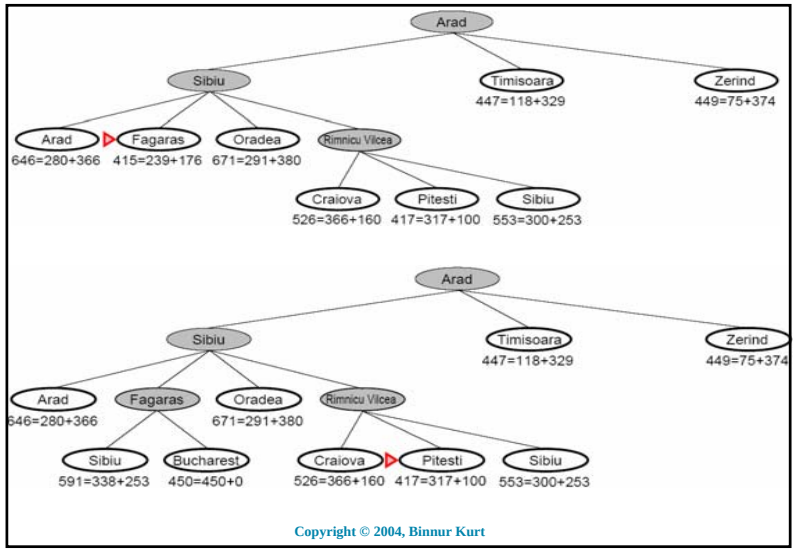


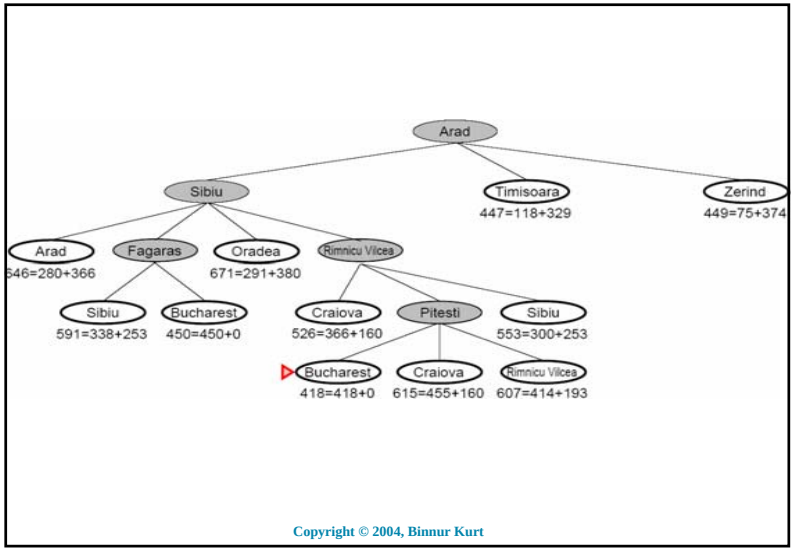


A* (A-star) Search

- Don't simply minimize the cost to goal…
  minimize the cost from start to goal…
  - *f(n)* = *g(n)* + *h(n)*
    - *g(n)* = cost to get to *n* from *start*
    - *h(n)* = cost to get from *n* to *goal*
- Select node from fringe that minimizes f(n)







A* is Optimal?

- A* can be optimal if h(n) satisfies conditions
  - h(n) never overestimates cost to reach the goal
    - it is eternally optimistic
    - called an ***admissible heuristic***
    - f(n) never overestimates cost of a solution through n
  - Proof of optimality?

A* is Optimal

- We must prove that A* will not return a suboptimal goal or a suboptimal path to a goal
  - Let G be a suboptimal goal node
    - f(G) = g(G) + h(G)
    - h(G) = 0 because G is a goal node
    - f(G) = g(G) > C* (because G is suboptimal)
  - Let n be a node on the optimal path
    - because h(n) does not overestimate
    - f(n) = g(n) + h(n) <= C*
  - Therefore f(n) <= C* < f(G)
    - node n will be selected before node G

Repeated States and GRAPH-SEARCH

- GRAPH-SEARCH always ignores all but the first occurrence of a state during search
  - Lower cost path may be tossed
    - So, don't throw away subsequent occurrences
    - Or, ensure that the optimal path to any repeated state is always the first one followed
  - Additional constraint on heuristic, consistency

Consistent heuristic: h(n)

- Heuristic function must be monotonic
  - for every node, *n*, and successor, *n′*, obtained with action *a*
    - estimated cost of reaching goal from n is no greater than cost of getting to n′ plus estimated cost of reaching goal from n′
    - h(n) <= c(*n*, *a*, *n′*) + h(*n′*)
  - This implies f(*n*) along any path are nondecreasing

Examples of consistent h(n)

- h(n) <= c(*n*, *a*, n^{succ}) + h(n^{succ})
  - recall h(*n*) is admissible
    - The quickest you can get there from here is 10 minutes
      - It may take more than 10 minutes, but not fewer
  - After taking an action and learning the cost
    - It took you two minutes to get here and you still have nine minutes to go
    - We cannot learn… it took you two minutes to get here and you have seven minutes to go

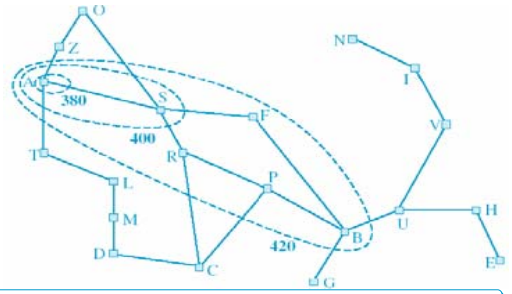
Proof of Monotonicity of f(*n*)

- If h(*n*) is consistent (monotonic) then f(*n*) along any path is nondecreasing
  - let n' be a successor of n
    - g(*n'*) = g(n) + c (n, a, *n'*) for some *a*
    - f(*n'*) = g(*n'*) + h(n')
      = g(n) + c(n, a, *n'*) + h(*n'*) >= g(n) + h(n) = f(n)

monotonicity implies
h(n) <= c(n, a, *n'*) + h(*n'*)

Contours

- Because f(*n*) is nondecreasing
- We can draw contours
  - If we know C*
  - We only need to explore contours less than C*



Properties of A*

- A* expands all nodes with f(n) < C*
- A* expands some (at least one) of the nodes on the C* contour before finding the goal
- A* expands no nodes with f(n) > C*
  - these unexpanded nodes can be **pruned**

A* is Optimally Efficient

- Compared to other algorithms that search from root
- Compared to other algorithms using same heuristic
- No other optimal algorithm is guaranteed to expand fewer nodes than A*
  (except perhaps eliminating consideration of ties at f(*n*) = C*)

Pros and Cons of A*

- A* is optimal and optimally efficient
- A* is still slow and bulky (space kills first)
  - Number of nodes grows exponentially with the length to goal
    - This is actually a function of heuristic, but they all have errors
  - A* must search all nodes within this goal contour
  - Finding suboptimal goals is sometimes only feasible solution
  - Sometimes, better heuristics are non-admissible

Local Search Algorithms and Optimization Problems

Characterize Techniques

- Uninformed Search
  - Looking for a solution where solution is a path from start to goal
  - At each intermediate point along a path, we have no prediction of value of path
- Informed Search
  - Again, looking for a path from start to goal
  - This time we have insight regarding the value of intermediate solutions

Now change things a bit

- What if the path isn't important, just the goal?
  - So the goal is unknown
  - The path to the goal need not be solved
- Examples
  - What quantities of quarters, nickels, and dimes add up to $17.45 and minimizes the total number of coins
  - Is the price of Microsoft stock going up tomorrow?

Local Search

- Local search does not keep track of previous solutions
  - Instead it keeps track of current solution (current state)
  - Uses a method of generating alternative solution candidates
- Advantages
  - Use a small amount of memory (usually constant amount)
  - They can find reasonable (note we aren't saying optimal) solutions in infinite search spaces

Optimization Problems

- Objective Function
  - A function with vector inputs and scalar output
    - goal is to search through candidate input vectors in order to minimize or maximize objective function
- Example

$$f(q,d,n) = \begin{cases} 10^6 & 0.25*q+0.1*d+0.005*n \neq 17.45 \\ q+d+n & \textit{otherwise} \end{cases}$$

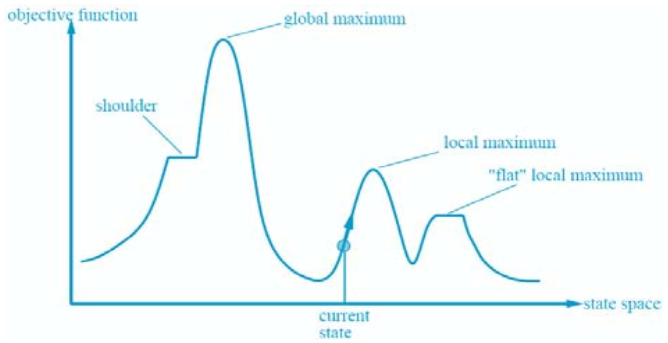
  - minimize *f*

Search Space

- The realm of feasible input vectors
  - Also called state-space landscape
  - Usually described by
    - number of dimensions (3 for our change example)
    - domain of each dimension ($\#_{quarters}$ is discrete from 0 to 69)
    - nature of relationship between input vector and objective function output
      - no relationship
      - smoothly varying
      - discontinuities

Search Space

- We are looking for global maximum (or minimum)



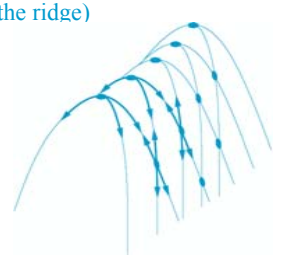
Hill Climbing

- Also called Greedy Search
  - Select a starting point and set current
  - evaluate (current)
  - loop do
    - neighbor = highest value successor of current
    - if evaluate (neighbor) <= evaluate (current)
      - return current
    - else current = neighbor

Hill climbing gets stuck

- Hiking metaphor (you are wearing glasses that limit your vision to 100 meters)
- Local maxima
  - Ridges

  (in cases when you can't walk along the ridge)
- Plateau
  - why is this a problem?



Hill Climbing Gadgets

- Variants on hill climbing play special roles
  - stochastic hill climbing
    - does not always choose the best successor
  - first-choice hill climbing
    - pick the first good successor you find
      - useful if number of successors is large
  - random restart
    - follow steepest ascent from multiple starting states
    - probability of finding global maximum increases with number of starts

Hill Climbing Usefulness

- It Depends
  - Shape of state space greatly influences hill climbing
  - local maxima are the Achilles heel
  - what is cost of evaluation?
  - what is cost of finding a random starting location?

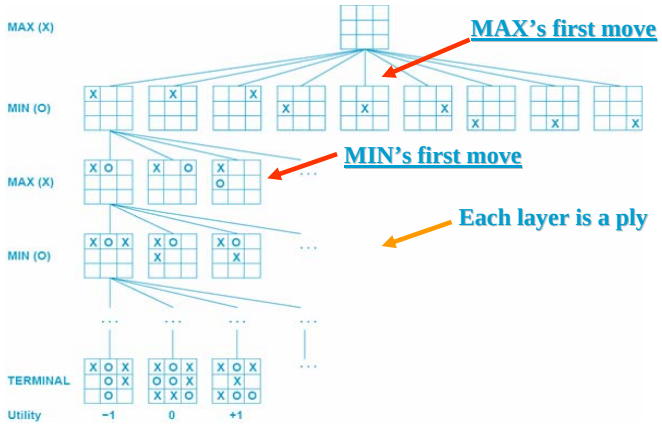
Adversarial Search: Optimal Search in Games

Chess Match–Spring 2003
ends in a 3-3 Draw

Games

- "Let's play a game?"
  - Tic-tac-toe

Tic-Tac-Toe game tree



What data do we need to play?

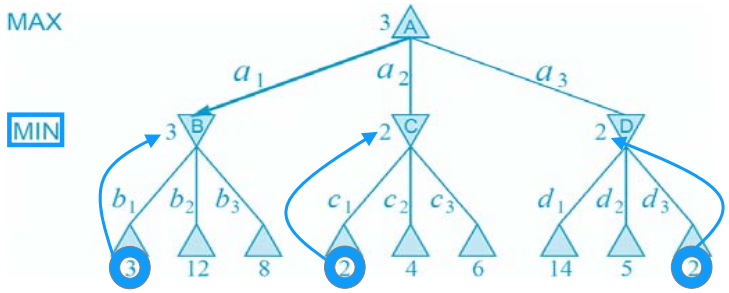
- Initial State
  - How does the game start?
- Successor Function
  - A list of legal (move, state) pairs for each state
- Terminal Test
  - Determines when game is over
- Utility Function
  - Provides numeric value for all terminal states

Minimax strategy

- Optimal Strategy
  - Leads to outcomes at least as good as any other strategy when playing an infallible opponent
  - Pick the option that minimizes the maximum damage your opponent can do
    - minimize the worst-case outcome
    - because your skillful opponent will certainly find the most damaging move

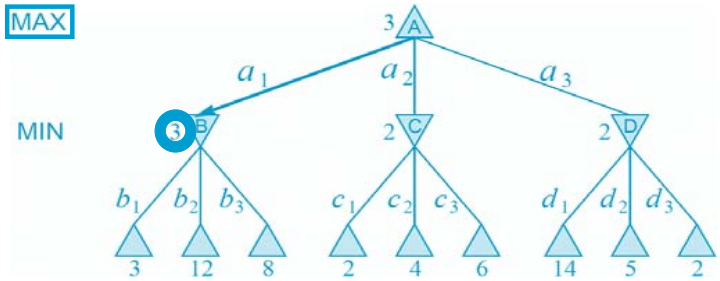
A two-ply example

- MIN considers minimizing how much it loses…



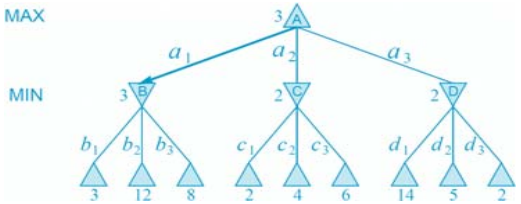
A two-ply example

- MAX considers maximizing how much it gains…



Minimax Algorithm

- We wish to identify minimax decision at the root
  - Recursive evaluation of all nodes in game tree



  - Time complexity = O (b^m)

Feasibility of minimax?

- How about a nice game of chess?
  - Avg. branching = 35 and Avg # moves = 50 for each player
    - $O(35^{100})$ time complexity = 10^{154} nodes
      - 10^{40} distinct nodes
- Minimax is impractical if directly applied to chess

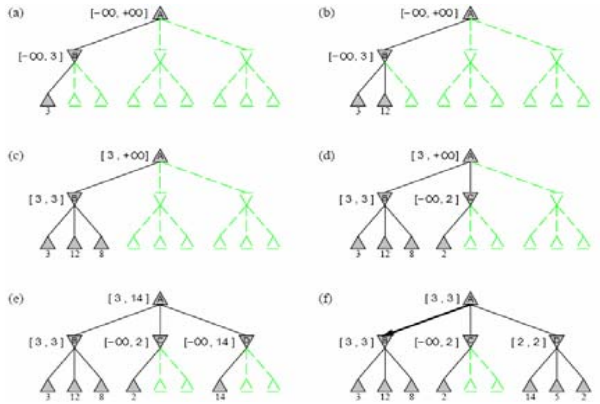
Pruning minimax tree

- Are there times when you know you need not explore a particular move?
  - When the move is poor?
  - Poor compared to what?
  - Poor compared to what you have explored so far

Alpha-beta pruning

- α
  - the value of the best (highest) choice so far in search of MAX
- β
  - the value of the best (lowest) choice so far in search of MIN
- Order of considering successors matters
  - If possible, consider best successors first

Alpha-beta pruning



Alpha-beta pruning

- Without pruning
  - $O(b^d)$ nodes to explore
- With a good heuristic pruner (consider part (f) of figure)
  - $O(b^{d/2})$
    - Chess can drop from $O(35^{100})$ to $O(6^{100})$
- With a random heuristic
  - $O(b^{3d/4})$

Real-time decisions

- What if you don't have enough time to explore entire search tree?
  - We cannot search all the way down to terminal state for all decision sequences
  - Use a heuristic to approximate (guess) eventual terminal state

Evaluation Function (Estimator)

- The heuristic that estimates expected utility
  - Cannot take too long (otherwise recourse to get answer)
  - It should preserve the ordering among terminal states
    - otherwise it can cause bad decision making
  - Define features of game state that assist in evaluation
    - what are features of chess?

Truncating minimax search

- When do you recurse or use evaluation function?
  - Cutoff-Test (state, depth) returns 1 or 0
    - When 1 is returned, use evaluation function

When do you cut off?

- When exploring beyond a certain depth
  - The horizon effect

When do you cut off?

- Cutoff if state is stable or quiescient (more predictable)

(b) White to move

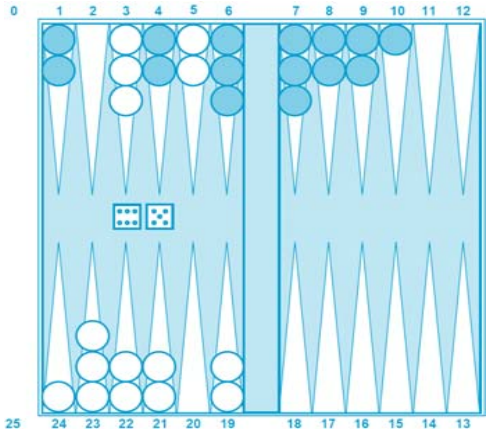
When do you cut off?

- Cutoff moves you know are bad (forward pruning)

Benefits of truncation

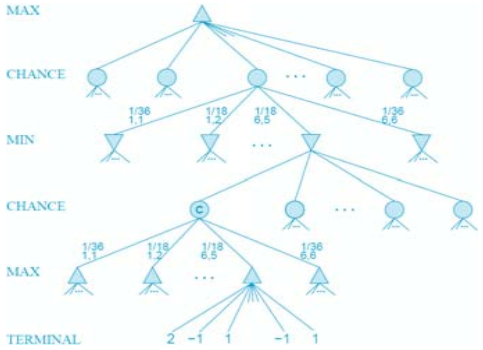
- Comparing Chess — Number of plys that can considered per unit time
  - Using minimax — 5 ply
  - Average Human — 6-8 ply
  - Using alpha-beta — 10 ply
  - Intelligent pruning — 14 ply

Games with chance: Backgammon



Games with chance

- How to include chance in game tree?
  - Add chance nodes



Expectiminimax

$$EXPECTIMINIMAX(n) = \begin{cases} UTILITY(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} EXPECTIMINIMAX(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} EXPECTIMINIMAX(s) & \text{if } n \text{ is a MIN node} \\ \sum_{s \in Successors(n)} P(s) \cdot EXPECTIMINIMAX(s) & \text{if } n \text{ is a chance node} \end{cases}$$

Pruning

- Can we prune search in games of chance?
  - Think about alpha-beta pruning
    - With alpha-beat, we don't explore nodes that we *know* are worse than what we know we can accomplish
    - With randomness, we never really what we will accomplish
      - chance node values are average of successors
- Thus it is hard to use alpha-beta

Building alpha-beta tree

- Can we restrict the size of game tree?
  - alpha-beta will blindly explore tree in depth-first fashion even if only one move is possible from root
  - even if multiple moves are possible, can we use a quick search to eliminate some entirely?
  - utility vs. time tradeoff to decide when to explore new branches or to stay with what you have

Metareasoning

- Reasoning about reasoning
  - alpha-beta is one example
    - think before you think
    - think about utility of thinking about something before you think about it
    - don't think about choices you don't have to think about

Goal-directed reasoning / planning

- Minimax starts from root and moves forward using combinatorial search
- What about starting at goal and working backward
  - We talked about difficulty of identifying goal states in bidirectional search
  - We do not know how to combine the two in practical way

Practice Session

A Water Jug Problem

- You are given two jugs, a 4lt one and a 3lt one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water.
- How can you get exactly 2lt of water into the 4lt jug?

The Missionaries and Cannibals Problem

- Three missionaries and three cannibals find themselves on one side of a river. They have agreed that they would all like to get to the other side. But the missionaries are not sure what else the cannibals have agreed to.
- So the missionaries want to manage the trip across the river in such a way that the number of missionaries on either side of the river is never less than the number of cannibals who are on the same side. The only boat available holds only two people at a time.

The Monkey and Bananas Problem

- A hungry monkey finds himself in a room in which a bunch of bananas is hanging from the ceiling. The monkey, unfortunately, cannot reach the bananas.
- However, in the room there are also a chair and a stick. The ceiling is just the right height so that a monkey standing on a chair could knock the bananas down with he stick.
- The monkey knows how to move around, carry other things around, reach for the bananas, and wave a stick in the air.
- What is the best of actions for the monkey to take to acquire lunch?