Machine Learning

State Space Search

State Space

State

A **state** of a search problem is an abstract representation of all relevant problem *features*.

A state can be viewed one possible step to take in the search.

- May not be necessary to examine each state
- May not be possible to get to each state

State Space

The **state space** of a problem is the set of all possible states of the problem.

A state space can be viewed as a directed graph where each state is a node and each edge is an action that can be performed.

A state space can be finite or infinite.

State Space Search

State space search is a process in which successive configurations or states of an instance are considered, with the intention of finding a goal state with a desired property.

Search Problems

A **search problem** can be defined formally by 5 components:

1. States
2. Initial State
3. Successor Function
4. Goal Test
5. Path Cost

Initial State

The **initial state** of a problem is where the search agent begins.

- Specially designated
- Can be viewed as the root of a search tree

Matt Johnson, Ph.D. 7

Successor Function

An *action* is how the agent moves between states.

A **successor function** applies each possible action to the current state

- Generates a set of reachable states
- May create states which have already been examined

Matt Johnson, Ph.D. 8

Path Cost

The **path cost** is the calculated weight of the edges connecting two nodes in the state space.

- For many simple problems this is just the number of nodes in the path
- Cost may be an estimation

The **step cost** of an action is the path cost between two adjacent nodes when performing an action.

Matt Johnson, Ph.D. 9

Goal Test

The **goal test** determines whether a given state is the goal state.

A **goal state** is a state that satisfies the goal test.

The goal test determines whether the current state:

- meets some abstract property
- is a member of an explicit state set

Matt Johnson, Ph.D. 10

8-Puzzle

State:

the location of tiles numbered 1 through 8 and the blank tile in a 3*3 grid

Initial State:

an arbitrary state in the state space

1	4	3
7		6
5	8	2

Matt Johnson, Ph.D. 11

8-Puzzle (2)

Successor Function:

legal states formed by moving tiles adjacent to the blank *up*, *down*, *left* and/or *right*

Goal Test:

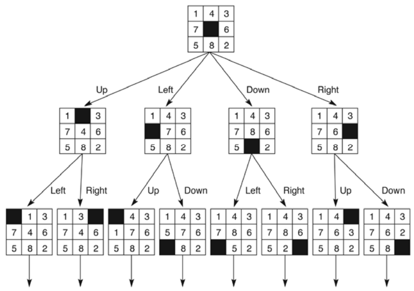
1	2	3
4	5	6
7	8	

Path Cost:

Number of moves taken

Matt Johnson, Ph.D. 12

8-Puzzle (3)



8-Queens

States:

Any arrangement of 0-8 queens on a chess board

Initial State:

An empty chess board

Successor Function:

All possible chess boards obtained by adding a queen to any empty space

8-Queens (2)

Goal Test:

A chess board with 8 queens with no queen able to “attack” another queen on the board

Path Cost:

Number of queens placed (irrelevant)

Routing Problem

States:

A location on the map, and the current time

Initial State:

Current location on map, time 0

Successor Function:

Location to every next intersection toward destination, current total time traveled to reach it.

Routing Problem (2)

Goal Test:

Desired address on map, shortest possible travel time.

Path Cost:

Total travel time to each intermediate state

Uniformed Search

Uniformed Search

In **uninformed search**, the algorithm has no additional information about states beyond that of the problem definition.

- Algorithm operates “blind”
- Can only generate successor states, and determine whether goal has been reached

Uninformed Search Algorithms

- Breadth-First Search
- Uniform-Cost Search
- Depth-First Search
- Depth-Limited Search
- Iterative-Deepening Search
- Bidirectional Search

Measuring Peformance

Completeness:

Is the algorithm guaranteed to find a solution?

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?

Complexity Metrics

The **branching factor** *b* is the maximum number of successors any node can have.

The **depth** ***d*** is the depth of the shallowest goal node.

The **maximum length** *m* is the maximum length of any path in the state space.

Terminology

Expanding a node (or **state**)

Applying the successor function to a node.

Known State

A state that has been generated by successor function

Implementation

All search algorithms need to maintain two lists

- **Closed list**
  - All states that have been expanded
  - Needed to avoid duplication
- **Frontier** (or **fringe** or **open list**)
  - All known states that have not yet been expanded
  - The ordering of the frontier varies depending on search algorithm

Breadth-First Search (BFS)

The frontier is maintained as a queue.

BFS is the search for the cautious.

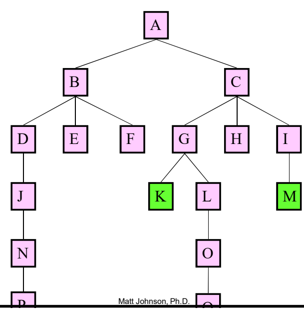
Completeness: Yes

Optimality: Yes

Time Complexity: $O(b^{d+1})$

Space Complexity: $O(b^{d+1})$

Breadth-First Search (2)



Uniform Cost Search

Frontier is sorted by *total* path cost to reach state.

When all path costs are 1, this is BFS

Completeness: Yes

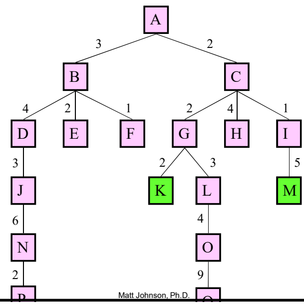
Optimality: Yes

Time Complexity: $O(b^{C^*/\varepsilon})$

Space Complexity: $O(b^{C^*/\varepsilon})$

C^* is cost of optimal solution, ε is minimum action cost.

Uniform Cost Search (2)



Depth-First Search (BFS)

The frontier is maintained as a stack.

DFS is the search for the brave.

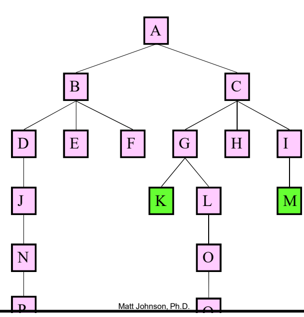
Completeness: No

Optimality: No

Time Complexity: O(b^m)

Space Complexity: O(bm)

Depth-First Search



Depth-Limited Search

DFS with a predetermined cutoff depth l.

Terminates once all nodes at l are generated.

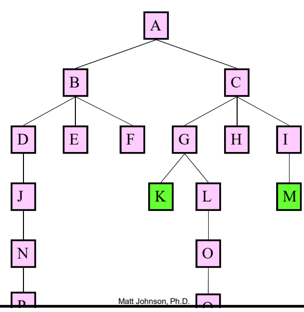
Completeness: No

Optimality: No

Time Complexity: $O(b^l)$

Space Complexity: $O(bl)$

Depth-Limited Search (2)



Iterative Deepening Search

Also known as Iterative Deepening DFS.

Depth-Limited Search with $l = 0, 1, 2, \ldots$

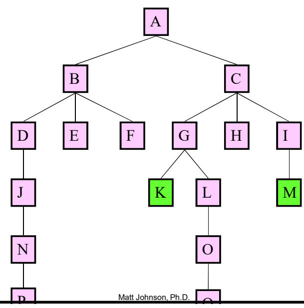
Completeness: Yes

Optimality: Yes

Time Complexity: $O(b^d)$

Space Complexity: $O(bd)$

Iterative Deepening Search (2)



Bidirectional Search

Two searches: one from start and one from goal.
Often BFS both ways, or BFS/DFS combo.

Completeness: Yes*
Optimality: Yes *
Time Complexity: $O(b^{d/2})$ *
Space Complexity: $O(b^{d/2})$*

*for BFS both ways.

Informed Search

Informed Search

Informed Search is a strategy that uses problem-specific knowledge beyond the problem definition itself to find a solution.

A key component of informed search algorithms is a ***heuristic function.***

Heuristics

A **heuristic function** ***h*****(*****x*****)** (also called simply a **heuristic**) is the estimated cost of the cheapest path from node *n* to any goal node.

If x is a goal node, $h(x) = 0$.

The heuristic is used to rank alternatives in a search algorithm in order to decide which branch to follow.

Best-First Search

In **Best-First Search**, the frontier is maintained as a priority queue.

- Traditionally, the node with the lowest priority is chosen for expansion.
- Implemented like other state space searches.

Best-First Search employs an **evaluation function** ***f*****(*****x*****)** to determine a node's priority.

Heuristic Search

The two main methods for Heuristic Search are:

1. Greedy Best-First Search
2. A^* Search

The difference between these two methods is the format of the evaluation function.

Greedy Best-First Search

In **Greedy Best-First Search**, the evaluation function is:

$$f(x) = h(x)$$

Only a heuristic function is used to prioritize nodes on the frontier.

Greedy Best-First Search (2)

As an example, consider the route finding problem. One straightforward heuristic to use would be

$h(x)$ = straight-line distance of city to goal

The heuristic is useful because it has a strong correlation to actual road distance to travel.

Greedy Best-First Search (3)

Greedy Best-First Search resembles depth-first search:

Completeness: No

Optimality: No

Time Complexity: O(b^m)

Space Complexity: O(bm)

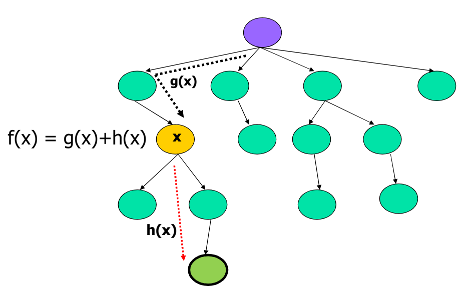
A* Search

In **A* Search**, the evaluation function is:

$$f(x) = g(x) + h(x)$$

- The function $g(x)$ is the total cost from the start node to node x
- The function $h(x)$ is the estimated total cost from node x to a goal node.
- $f(x)$ is the therefore the estimated cost of the cheapest solution through x.

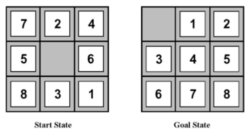
A* Search (2)



A^* Search (3)

For the 8-puzzle, two possible heuristics would be:

1. The number of tiles out of place.
2. The Manhattan distance of all tiles. The **Manhattan distance** of a tile is the number of actions necessary to move the tile to its correct goal position.



A^* Search (4)

A heuristic is **admissible** if it never overestimates the total cost to reach the goal.

A heuristic is **consistent** if, for every node x and every successor x' of x,

$$h(x) \leq c(x) + h(x')$$

where $c(x)$ is actual cost of reaching x' from x.

A^* Search (5)

Pruning is the process whereby a node is not expanded because its $f(x)$ value is higher than other choices at higher depth.

Pruning can make A^* Search more efficient than uninformed search methods.

A^* Search (6)

Completeness: Yes, if the heuristic is admissible.
Optimality: Yes, if the heuristic is consistent.
Time Complexity: $O(b^{d+1})$ without pruning
Space Complexity: $O(b^{d+1})$ without pruning

Games

Why Games?

Games typically have:

- Small well-defined set of rules
- Well-defined set of knowledge
- Easy to evaluate performance
- Large state spaces
- Too large for exhaustive search methods

Game Theory

Mathematical Game Theory is a branch of Economics.

Example: Prisoners' Dilemma

John Nash, early 1950's

The Nash equilibrium is a decision-making theorem within game theory that states a player can achieve the desired outcome by not deviating from their initial strategy.

Game Theory (2)

Example: Prisoners' Dilemma

Two members of a criminal gang are arrested. The prosecutors offer each prisoner a bargain.

- If A and B each betray the other, each of them serves two years in prison
- If A betrays B but B remains silent, A will be set free and B will serve three years in prison (and vice versa)
- If A and B both remain silent, both of them will serve only one year in prison (on the lesser charge)

AI Games

In typical AI games

- There are 2 players
- Actions alternate between players
- Utility functions are equal but opposite
  - A good outcome from Player A's perspective is a bad outcome from Player B's
  - opposition between the agents' utility functions makes the situation *adversarial*

Games vs. Search

Unlike standard state space search, in adversarial search:

- Actions taken by opponents are unpredictable
- The solution to the search is a game strategy
- There may be time limits
- It's unlikely to find a direct path to goal

Adverserial Search

Adversarial Search

Adversarial Search is a state space search algorithm for competitive environments

- Each potential board or game position is a state
- Each possible action is a move to another state
- The state space can be HUGE!!!!!!!
- Large branching factor (about 35 for chess)
- Terminal state could be deep (about 50 for chess)

Adversarial Search (2)

Initial State:

Initial board or game configuration

Successor Function:

All legal next actions and the resultant states

Terminal Test:

Whether the game has entered a terminal state and game is won or lost

Utility Function:

The numeric value for game outcomes

e.g. win (1), lose (-1), draw (0)

Game Trees

The game tree is the search space

- Each complete path represents one possible game
- Not all paths are examined

The levels of the tree indicate a player's possible moves:

- The root of the tree is the initial state
- Next level is all of AI's moves
- Next level is all of opponent's moves

Plies

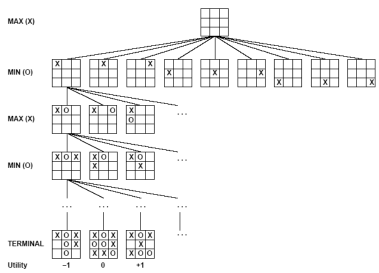
A **ply** is one turn of a game:

The **ply depth** is how many plies down the game tree are examined when choosing an action.

Example: Tic-Tac-Toe

- Root has 9 blank squares
- Ply 1: has 8 blank squares
- Ply 2: has 7 blank squares
- Ply 3: has 6 blank squares

Tic-Tac-Toe



Minimax Search

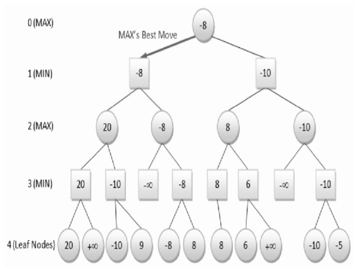
Minimax Search is a competitive search algorithm for two players agents called MAX and MIN:

- MAX
  - Higher utility is better
  - Starts game
- MIN
  - Lower Utility is better

Minimax Search (2)

```
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)
```

Minimax Search Example



Ply Limit

It may not be computationally feasible to search entire game tree to all win/lose/draw.

You can instead use a **ply limit** in your search, then heuristically evaluate nodes at the limit. For Tic-Tac-Toe example, :

- $h(x) = \infty$ for win
- $h(x) = -\infty$ for loss
- $h(x)$ = # of ways left to win - # of ways left to lose otherwise

Metrics

Complete: Yes if the tree is finite

Optimal: Yes, against an optimal opponent

Time Complexity: $O(b^m)$

Space Complexity $O(bm)$

Alpha-Beta Pruning

Pruning

Pruning is the process whereby a node is not expanded because its $f(x)$ value is higher/lower than other choices at higher depth.

- The problem with minimax search is that the number of game states it has to examine is exponential in the number of moves.
- Pruning can be used to reduce the time complexity of Minimax Search!

Alpha-Beta Pruning

Alpha-Beta Pruning remembers two values:

- **α** is the value of the best choice we have found so far for MAX at any choice point along the path
- **β** is the value of the best choice we have found so far for MIN at any choice point along the path

Minimax Search can be modified to:

- prune values lower than α when maximizing
- prune values higher than β when minimizing

Alpha-Beta Pruning (2)

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

---

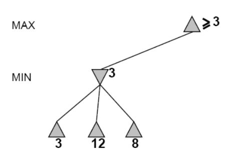
function MAX-VALUE(*state, game*, α, β) **returns** the minimax value of *state*
  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each *s* **in** SUCCESSORS(*state*) **do**
    $\alpha \leftarrow \max(\alpha, \text{MIN-VALUE}(s, game, \alpha, \beta))$
    if $\alpha \geq \beta$ **then return**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

---

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

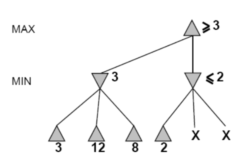
Alpha-Beta Pruning Example

Step 1:



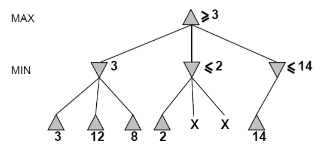
Alpha-Beta Pruning Example (2)

Step 2:



Alpha-Beta Pruning Example (3)

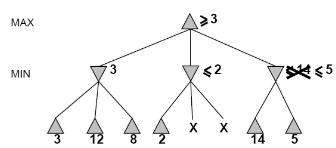
Step 3:



Matt Johnson, Ph.D. 73

Alpha-Beta Pruning Example (4)

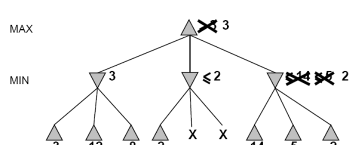
Step 4:



Matt Johnson, Ph.D. 74

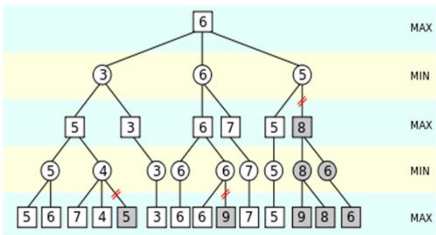
Alpha-Beta Pruning Example (5)

Step 5:



Matt Johnson, Ph.D. 75

Alpha-Beta Pruning Example 2



Alpha-Beta Pruning Results

- Pruning *<u>does not</u>* affect outcome.
- Effectiveness depends on order of successors.
- If best successor is evaluated first, the search is O($b^{d/2}$) instead of O(b^d).
  - This means that in the same amount of time, Alpha-Beta search can search twice as deep!
  - Can easily reach depth of 8 and play good chess (with a branching factor of ~6 instead of 35)

Games of Chance

Nondeterministic Games

A **Game of Chance** is a game with a nondeterministic environment.

- Chance is introduced by dice, card shuffling, etc.
- Probabilities of outcomes must be indicated in the game tree.

A **chance node** is a state in the game tree which is assigned a probability.

Expected Value

The **expected value** of on outcome is:

$$\sum_{i=0}^{k} x_i p_i$$

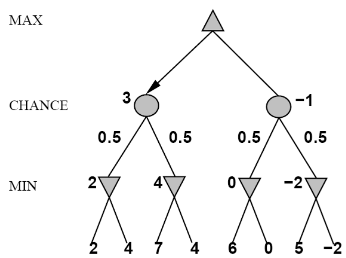
where x_i is the utility/heuristic value of each outcome and p_i is its chance of occurring.

Expectiminimax

Expectiminimax is a minimax search that handles chance nodes:

```
if state is a MAX node then
   return highest Expectiminimax
if state is a MIN node then
   return lowest Expectiminimax
if state is a CHANCE node then
   return weighted average of Expectiminimax
```

Expectiminimax (2)



Matt Johnson, Ph.D.

82