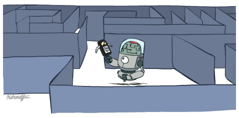
Artificial Intelligence

Module 3: Search Strategies

PART 3.3: Informed Search



Dr. Chandra Prakash

Assistant Professor

Department of Computer Science and Engineering

(Slides adapted from StuartJ. Russell, B Ravindran, Mausam, Prof. Pallab Dasgupta, Prof. Partha Pratim Chakrabarti, Saikishor Jangiti

Module 3: Search Strategies

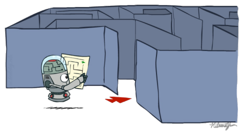
- PART 3.1: Search
- PART 3.2: Uninformed Search
- PART 3.3: Informed/Heuristic Search
  - Heuristics
  - Best First Search/ Greedy Search
  - A* Search
- PART 3.4: Beyond Classical Search:
  - Local Search
- PART 3.5: Constraint Satisfaction Problems
- PART 3.6: Adversarial Search

Recap: Search

- Important part of intelligence is to
  - Try to solve a new problem
- Develop a general purpose algorithm that can solve any kind of problem
  - general purpose representation for the problem
- Many problem in the world can be formated as Search Problem
- If only Input and Output is given
  - How to solve ??
    - Search Algorithms

Recap: Search

- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test



- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)
- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

Recap BASIC ALGORITHMS:DFS, IDS, BFS

1. [Initialize]

Initially the OPEN List contains the Start Node s. CLOSED List is Empty.

2. [Select]

Select the first Node n on the OPEN List. If OPEN is empty, Terminate

3. [Goal Test]

If n is Goal, then decide on Termination or Continuation / Cost Updation

4. [Expand]

a) Generate the successors n_1, n_2, …. n_k, of node n, based on the State Transformation Rules

b) Put n in LIST CLOSED

c) For each n_i, not already in OPEN or CLOSED List, put n_i in the FRONT (for DFS) / END (for BFS) of OPEN List

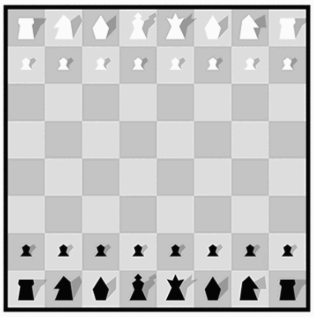
d) For each n_i already in OPEN or CLOSE decide based on cost of the paths

5. [Continue]

Go to Step 2

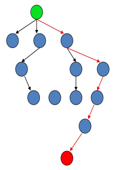
- Algorithm IDS Performs DFS Level by Level Iteratively (DFS (1), DFS (2), ……. and so on)

What you think ?



Intuition

- “Intuition, like the rays of the sun, acts only in an inflexibly straight line; it can guess right only on condition of never diverting its gaze; the freaks of chance disturb it.”
  — Honoré de Balzac
- Intuition -
  - Right /Wrong (replan)
  - Smart about what paths to try
- **Blind Search Vs Informed Search**
  - What is the difference ??
    - By systematically generating new states and testing against the goal.
    - By using problem-specific knowledge to find solutions more efficiently
  - How do we formally specify this ?
    - a node is selected for expension based on an evaluation function that estimates cost to goal.
    - evaluation function



General Tree/Graph Search Paradigm

```
function TREE-SEARCH(problem, fringe) return 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, STATE[node]) then return node
        for child-node in EXPAND(STATE[node], problem) do
            fringe ← INSERT(child-node, fringe)
        end
    end
```

open list/ fringe/ frontier

```
function GRAPH-SEARCH(problem, fringe) return 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
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

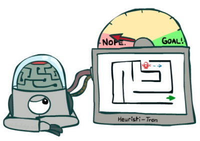
Informed Search Strategies

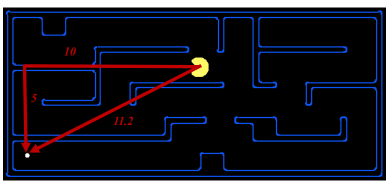
Use heuristic knowledge to incraese efficiency of search:

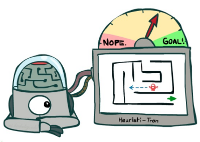
- **Select which node to expand next during search**
- While expanding a node decide which successors to generate and which to ignore
- Remove from the search space some nodes that have previously been generated – prune the search space
- covers techniques (some developed outside of AI) that don’t try to cover the whole space and only the goal state, not the steps, are important.

Search Heuristics

- A heuristic is:
  - A function that estimates how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing
    - Manhattan $(x_1, y_1, x_2, y_2) = |x_1 - x_2| + |y_1 - y_2|$
    - Euclidean $(x_1, y_1, x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$







Heuristics

- Heuristic is derived from heuriskein in Greek, meaning “to find” or “to discover”
- The term heuristics is often used to describe
  - rules of thumb or advice
  - that are generally effective, but are not guaranteed to work in every case.
- In the context of search, a heuristic is a function that
  - takes a state as an argument
  - and returns a number that is an estimate of the merit of the state with respect to the goal.
- Heuristics are like tour guides
- They are good to the extent that they
  - point in generally interesting directions;
- They are bad to the extent that they
  - may miss points of interest to particular individuals.
- On the average they improve the quality of the paths that are explored.
- Special purpose heuristics exploit domain specific knowledge

Heuristics

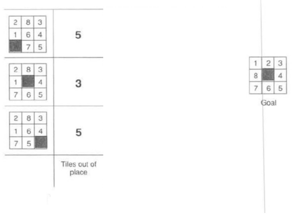
- A heuristic algorithm
  - improves the average-case performance,
  - does not necessarily improve the worst-case performance.
- Not all heuristic functions are beneficial.
  - The time spent evaluating the heuristic function in order to select a node for expansion must be recovered by a corresponding reduction in the size of the search space explored.
  - Useful heuristics should be computationally inexpensive!
- Well designed heuristic functions can play an important part in efficiently guiding a search process toward a solution.

Example Simple Heuristic functions

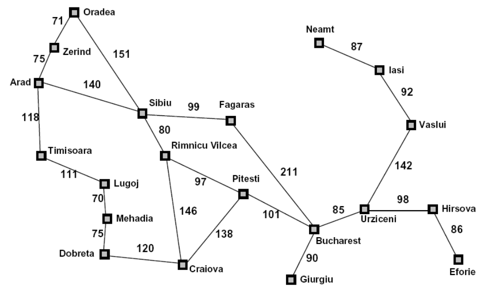
- Chess : The material advantage of our side over opponent.
- TSP: the sum of distances so far
- Tic-Tac-Toe: ???
- 4 square problem :

- 8 square problem :

Possible Heuristic Way for 8-Puzzel



Example: Heuristic Function



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

h(x)

Generate-and-Test

Example: coloured blocks

"Arrange four 6-sided cubes in a row, with each side of each cube painted one of four colours, sucl that on all four sides of the row one block face of each colour is showing."

- Heuristic: if there are more red faces than other colours then, when placing a block with several red faces, use few of them as possible as outside faces.
- Heuristic generate-and-test:
  - not consider paths that seem unlikely to lead to a solution.

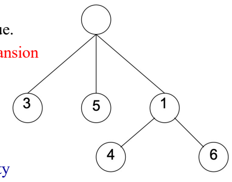
Greedy Search / Best-First Search / Greedy best-first Search

Greedy Search /Best-First Search

- Combines the advantages of both DFS and BFS into a single method.
  - Depth-first search: not all competing branches having to be expanded.
  - Breadth-first search: not getting trapped on dead-end paths.
- Combining the two is to follow a single path at a time, but
  - switch paths whenever some competing path look more promising than the current one.
  - At each step of the BFS search process, we select the most promising of the nodes we have generated so far.
  - This is done by applying an appropriate heuristic function to each of them.
  - We then expand the chosen node by using the rules to generate its successors
- This is called OR-graph, since each of its branches represents an alternative problem solving path

Greedy Best-First Search

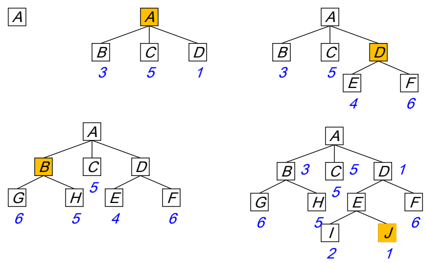
- An instance of the general Tree Search.
- Use an evaluation function f(n) for node n.
- Always choose the node from fringe that has the lowest f value.
- A search strategy is defined by picking the order of node expansion
- Idea: use an evaluation function f(n) for each node
  - estimate of "desirability“
  - Expand most desirable unexpanded node
- Implementation:
  - Order the nodes in fringe in decreasing order of desirability
  - Can be implemented via a priority queue that will maintain the fringe in ascending order of f-values
- Special cases:
  - Greedy Best-First Search (or Greedy Search)
  - A* Search



Greedy Best-First Search

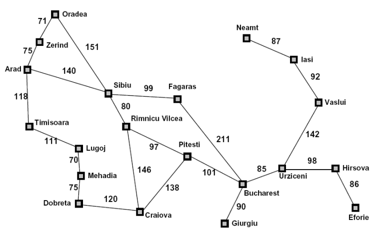
- Heuristic search uses problem-specific knowledge: evaluation function.
- Choose the seemingly-best node based on some estimate of the cost of the corresponding solution.
- Need estimate of the cost to a goal
  - e.g. Depth of the current node
    - Sum of the distances so far
    - Euclidean distance to goal etc.
- Heuristics: rules of thumb
- Goal: to find solutions more efficiently
- **Heuristic function**
  - h(n) = estimated cost of the cheapest path from node n to a goal node.
  - (h(n) = 0, for a goal node)

Greedy Best-First Search



Example: Heuristic Function

- h(n) = estimated best cost to goal from n
- h(n) = 0 if n is a goal

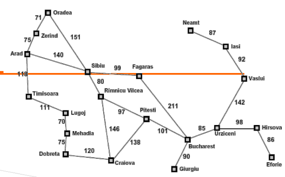


h(n)

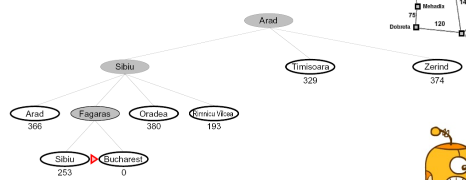
Straight−line distance to Bucharest

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

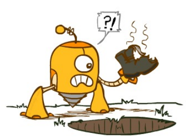
Greedy Best-First Search



- Expand the node that seems closest…

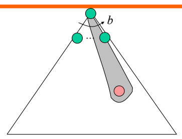


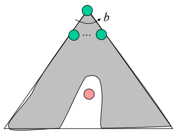
- What can go wrong?



Greedy Best-First Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state
- A common case:
  - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS



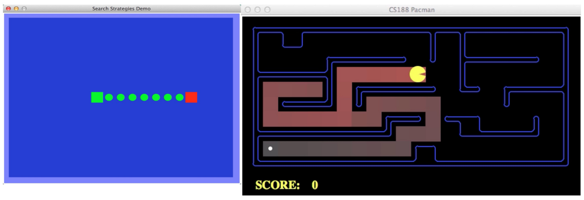


Old (Uninformed) Friends

- Breadth First = Best First Search
  - with f(n) = depth(n)
- Uniform cost search =Best First Search
  - with f(n) = the sum of edge costs from start to n

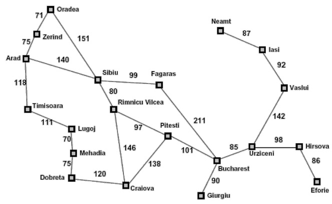
g(n)

Video of Demo Contours Greedy Best-First Search (Empty and Pacman Small Maze)



Greedy best-first search

- Evaluation function f(n) = h(n) (heuristic function)
  = estimate of cost from n to goal
  - e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that appears to be closest to goal



Properties of greedy best-first search

- Complete?
  - No – can get stuck in loops, e.g., Iasi → Neamt →Iasi →Neamt
- 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

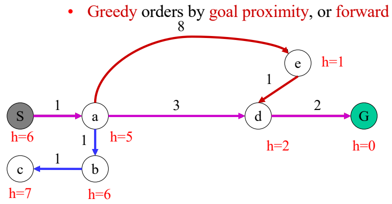
A* Search

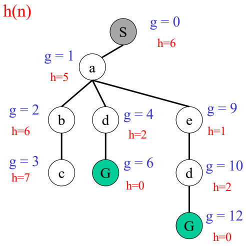
- (Hart et al., 1968):
- Idea: avoid expanding paths that are already expensive
- Evaluation function f(n) = g(n) + h(n)
  - g(n) = cost so far to reach n
  - h(n) = estimated cost from n to goal
- f(n) = estimated total cost of path through n to goal

A*

Combining UCS and Greedy

- Uniform-cost orders by path cost, or backward cost : g(n)
  - how far I am from Initial Position . g(n) ?????
- Greedy orders by goal proximity, or forward cost : h(n)





- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

ALGORITHM A* (BEST FIRST SEARCH IN OR GRAPHS)

Each Node n in the algorithm has a cost g(n) and a heuristic estimate h(n), f(n) = g(n) + h(n).
Assume all c(n,m) > 0

1. [Initialize] Initially the OPEN List contains the Start Node s. g(s) = 0, f(s) = h(s). CLOSED List is Empty.

2. [Select] Select the Node n on the OPEN List with minimum f(n). If OPEN is empty, Terminate with Failure

3. [Goal Test, Terminate] If n is Goal, then Terminate with Success and path from s to n.

4. [Expand]

a) Generate the successors n_1, n_2, …. n_k, of node n, based on the State Transformation Rules

b) Put n in LIST CLOSED

c) For each n_i, not already in OPEN or CLOSED List, compute

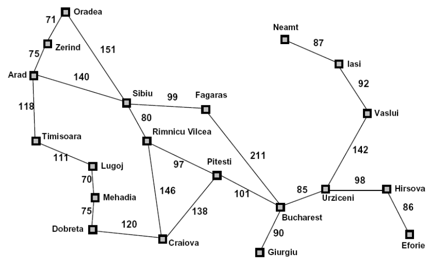
- g(n_i) = g(n) + c(n, n_i), f(n_i) = g(n_i) + h(n_i) , Put n_i in the OPEN List

d) For each n_i already in OPEN, if g(n_i) > g(n) + c(n,n_i), then revise costs as:

- g(n_i) = g(n) + c(n, n_i), f(n_i) = g(n_i) + h(n_i)

5. [Continue] Go to Step 2

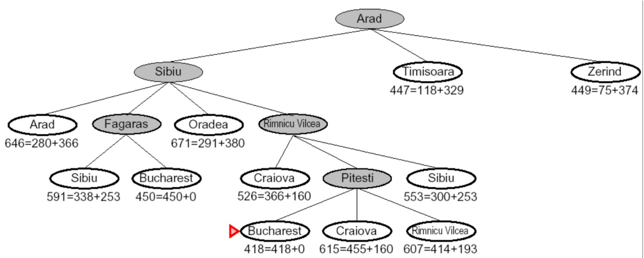
Example: Heuristic Function



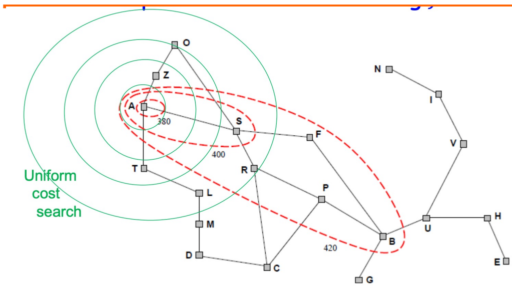
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

A* for Romanian Shortest Path



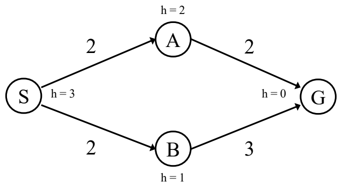
A∗ search: f -contours



A∗ gradually adds " f-contours" of nodes

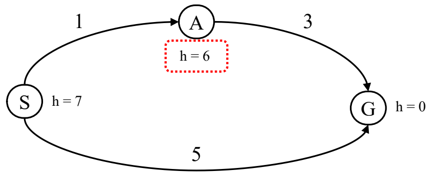
When should A* terminate?

- Should we stop when we enqueue/open-list a goal?



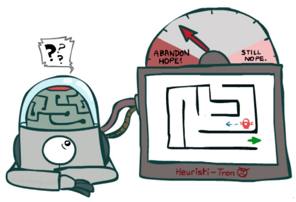
- No: only stop when we dequeue a goal
- remove from fringe/ close list.

Is A* Optimal?

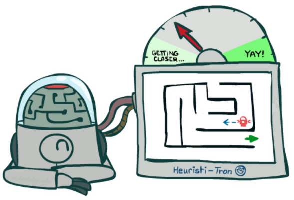


- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

Idea: Admissibility / Admissible Heuristics



Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe



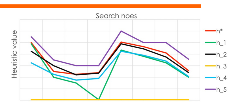
Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

Admissible Heuristics

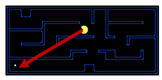
- A heuristic h is admissible (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal from n

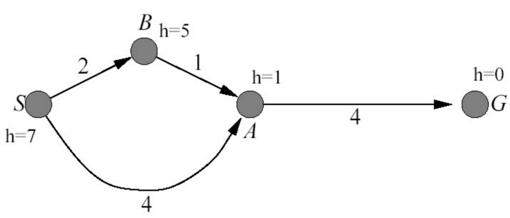


- Examples:



  - Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Coming up with admissible heuristics is most of what's involved in using A* in practice.

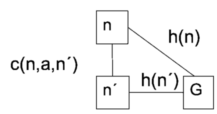
Example



Source: *http://stackoverflow.com/questions/25823391/suboptimal-solution-given-by-a-search*

Consistent Heuristics

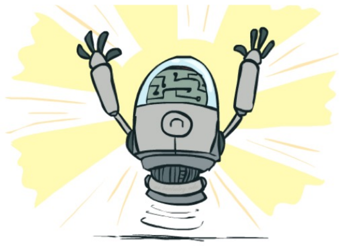
- h(n) is consistent if
  - for every node n
  - for every successor n´ due to legal action a
  - h(n) <= c(n,a,n´) + h(n´)



- Every consistent heuristic is also admissible.
- Theorem: If h(n) is consistent, A* using GRAPHSEARCH is optimal

Optimality of A* Tree Search

Theorem: If h(n) is admissible, A* using TREE-SEARCH is optimal



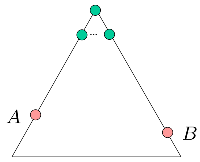
Proof of Optimality of A* Tree Search

Assume:

- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim:

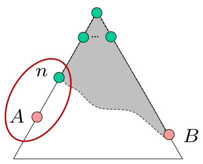
- A will exit the fringe before B



Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
  1. f(n) is less or equal to f(A)

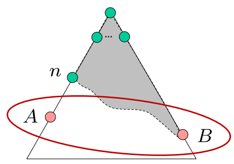


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)

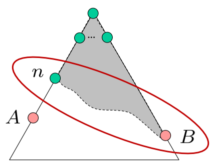


$g(A) < g(B)$	B is suboptimal
$f(A) < f(B)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor n of A is on the fringe, too (maybe A!)
- Claim: n will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. n expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal



$$f(n) \leq f(A) < f(B)$$

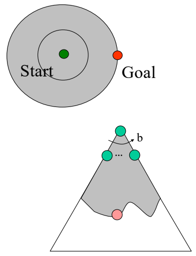
Properties of ALGORITHM A*

- Complete?
  - Yes (unless there are infinitely many nodes with f ≤ f(G))
- Time?
  - Exponential (worst case all nodes are added)
- Space?
  - Keeps all nodes in memory
- Optimal?
  - Yes (depending upon search algo and heuristic property)

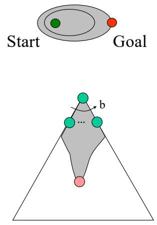
- If heuristic estimates are non-negative, Lower bounds and edge costs are positive:
  - first solution is optimal
  - no node in closed in ever reopened
  - whenever a node is removed from open its Minimum cost from start is found
  - every node n with f(n) less than optimal Cost is expanded
  - if heuristics are more accurate then Search is less

UCS vs A* Contours

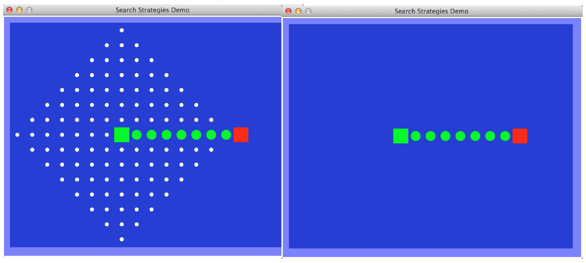
- Uniform-cost expands equally in all "directions"



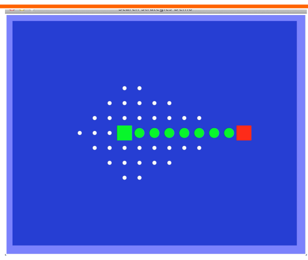
- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



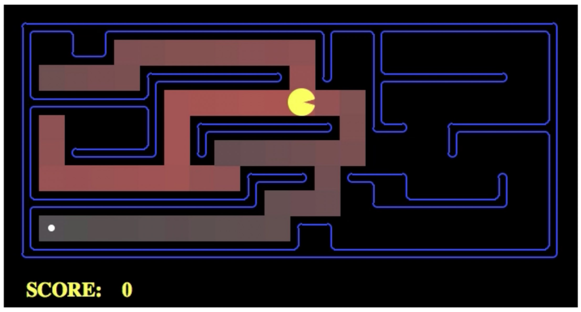
Video of Demo Contours (Empty) UCS or Best First Search Greedy



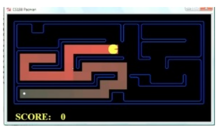
Video of Demo Contours (Empty) – A*



Video of Demo Contours (Pacman Small Maze) – A*

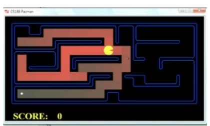


Comparison



Greedy

Uniform Cost

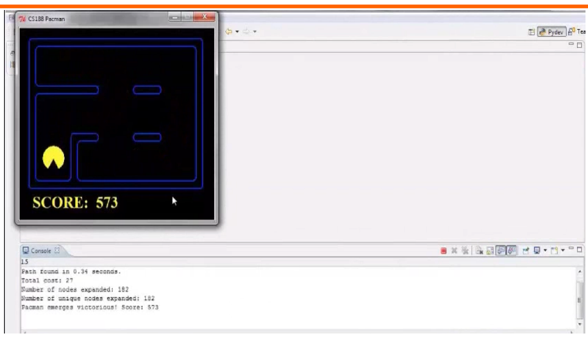


A*

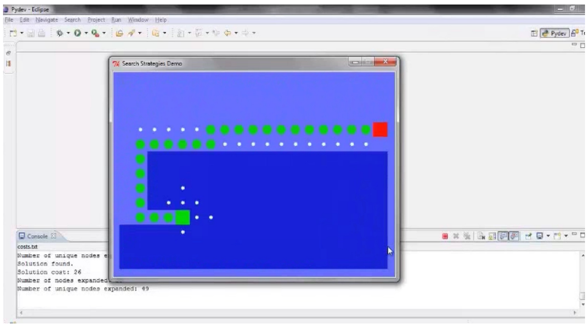
A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Puzzle Solving:
- Terrain Exploration for Drones
- Medical Image Processing:
- …

Video of Demo Pacman (Tiny Maze) – UCS / A*

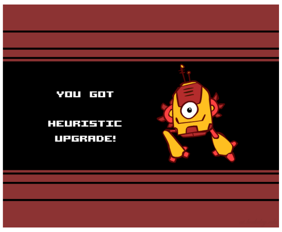


Video of Demo Empty Water Shallow/Deep – Guess Algorithm



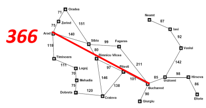
Creating Heuristics

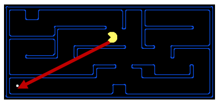
- How domain designer come up with Heuristic function



Creating Admissible Heuristics

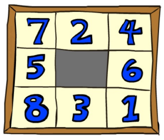
- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to *relaxed problems*, where new actions are available



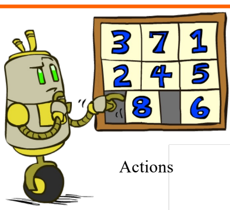


- Inadmissible heuristics are often useful too

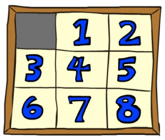
Example: 8 Puzzle



Start State



Actions

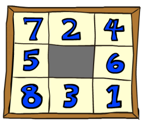


Goal State

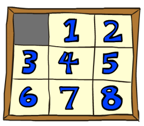
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

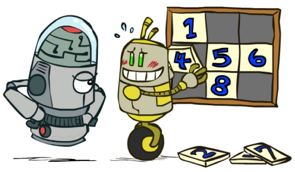
- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = *8*
- This is a *relaxed-problem* heuristic



Start State



Goal State

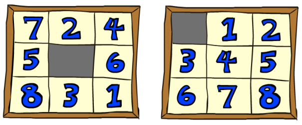


	Average nodes expanded when the optimal path has…			
Depth	…4 steps	…8 steps	…12 steps	… 24
UCS	112	6,300	3.6×10^6	too many
TILES	13	39	227	39,135 nodes

Statistics from Andrew Moore

8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why is it admissible?
- h(start) = *3 + 1 + 2 +2+2+3+3+2 = 18*



Start State Goal State

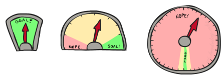
	Average nodes expanded when the optimal path has…			
	…4 steps	…8 steps	…12 steps	24 depth
TILES	13	39	227	39,135 nodes
MANHATTAN	12	25	73	1,641 nodes

Dominance and Relaxed problems

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
  - then h_2 dominates h_1
  - h_2 is better for search
- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
  - If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
  - If the rules are relaxed so that a tile can move to any adjacent square, then h_2 (n) gives the shortest solution

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?
    - We need to compute search problem within a search problem



- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

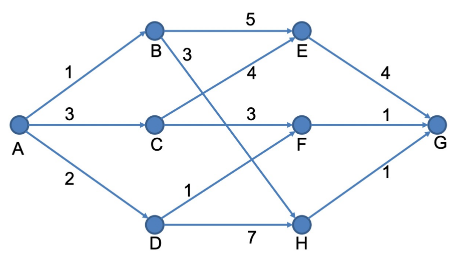
Variants in A*

- **Limitation of A***
  - Memory problem ??
- **Think what to do ?**
  - Overcome the space problem of A*, without sacrificing optimality or completeness
  - **Play with f(n)**
    - Weighted A* expands states in the order of f = g+εh values, where ε > 1 biases towards states that are closer to the goal.
  - **Cutoff**

Memory-bounded heuristic search

- **Iterative Deepening A* (IDA*)**
  - a logical extension of Iterative Deepening search (IDS) to use heuristic
  - While (solution not found)
    - the cutoff used is the f-cost (g+h) rather than the depth
    - Do DFS but prune when cost (f) > current bound
    - Increase bound
- **Depth First Branch and Bound**
  - example : Game playing
  - 2 mechanisms:
    - BRANCH: A mechanism to generate branches when searching the solution space
      - Heuristic strategy for picking which one to try first.
    - BOUND: A mechanism to generate a bound so that many branches can be terminated

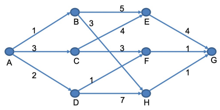
Example

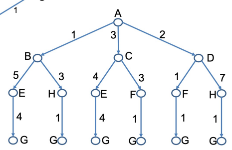


Find optimal path from A to G

DEPTH FIRST BRANCH AND BOUND (DFBB)

- Depends on branching factor
  - E.g., Branch policy: take lowest cost edge first

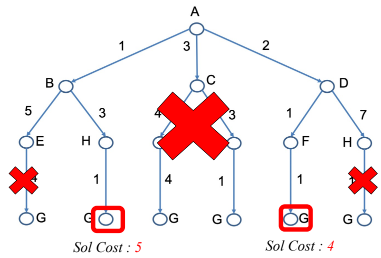




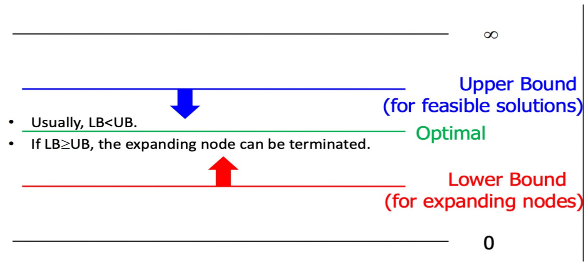
DEPTH FIRST BRANCH AND BOUND (DFBB)

1. Initialize Best-Cost to INFINITY
2. Perform DFS with costs and Backtrack from any node n whose f(n) ≥ Best-Cost
3. On reaching a Goal Node, update Best-Cost to the current best
4. Continue till OPEN becomes empty

DFS B&B



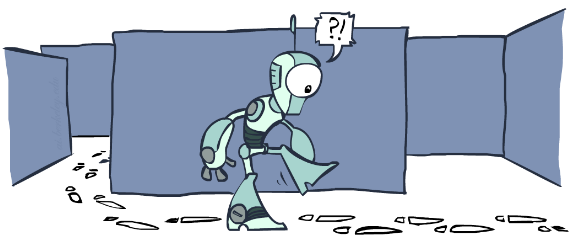
For Minimization Problems



DFS B&B vs. IDA*

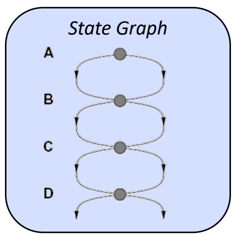
- Both optimal
- IDA* never expands a node with f > optimal cost
  - But not systematic
- DFb&b systematic never expands a node twice
  - But expands suboptimal nodes also
  - we should have domain knowledge for better upper bound
- Search tree of bounded depth?
- Easy to find suboptimal solution?
- Infinite search trees?
- Difficult to construct a single solution?

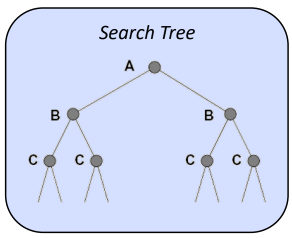
Graph Search



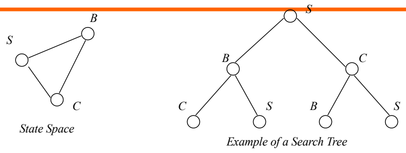
Tree Search: Extra Work!

- Repeated states
- Failure to detect repeated states can cause exponentially more work.





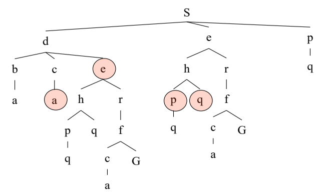
Solutions to Repeated States



- Method 1 ← *suboptimal but practical*
  - do not create paths containing cycles (loops)
- Method 2 ← *optimal but memory inefficient*
  - never generate a state generated before
    - must keep track of all possible states (uses a lot of memory)
    - e.g., 8-puzzle problem, we have 9! = 362,880 states

Graph Search

- In BFS, for example, we shouldn't bother expanding the circled nodes (why?)

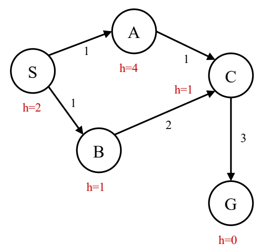


Graph Search

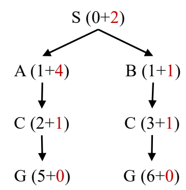
- Idea: Never expand a state twice
- How to implement:
  - Tree search + set of expanded states ("closed set")
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set
- Important: store the closed set as a set, not a list
- Can graph search wreck completeness? Why/why not?
- How about optimality?

A* Graph Search Gone Wrong?

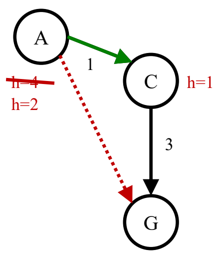
State space graph



Search tree



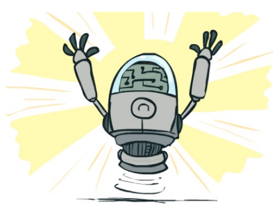
Consistency of Heuristics



- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal
    - h(A) ≤ actual cost from A to G
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc
    - h(A) – h(C) ≤ cost(A to C)
- Consequences of consistency:
  - The f value along a path never decreases
    - h(A) ≤ cost(A to C) + h(C)
  - A* graph search is optimal

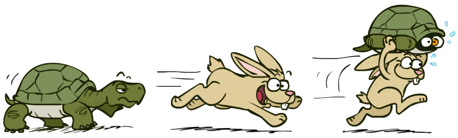
Optimality

- Tree search:
  - A* is optimal if heuristic is admissible
  - UCS is a special case (h = 0)
- Graph search:
  - A* optimal if heuristic is consistent
  - UCS optimal (h = 0 is consistent)
- Consistency implies admissibility
- In general, most natural admissible heuristics tend to be consistent, especially if from relaxed problems

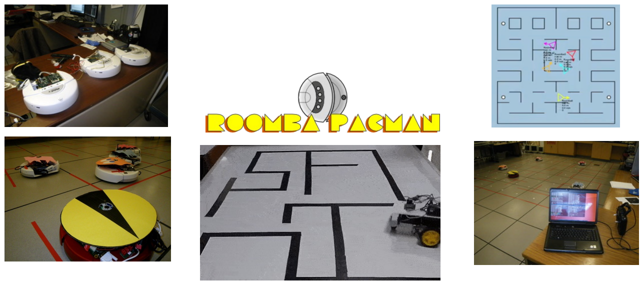


A*: Summary

- A* uses both backward costs and (estimates of) forward costs
- A* is optimal with admissible / consistent heuristics
- Heuristic design is key: often use relaxed problems



Pacman: Beyond Simulation?



Students at Colorado University: *http://pacman.elstonj.com*

Local Search and Optimization

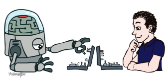
- Local Search and Optimization
- Previously: systematic exploration of search space.
  - Path to goal is solution to problem
- YET, for some problems path is irrelevant.
  - e.g 8-queens
- Different algorithms can be used
  - Local search

Module 3: Search Strategies

- Module 3: Search Strategies
  - PART 3.1: Search
  - PART 3.2: Uninformed Search
  - PART 3.3: Informed/Heuristic Search
    - Heuristics
    - Best First Search/ Greedy Search
    - A* Search
  - PART 3.4: Beyond Classical Search
    - Local Search
    - Problem reduction
  - PART 3.5: Constraint Satisfaction Problems
  - PART 3.6: Adversarial Search

References

- *Stuart Russell, Peter Norvig, Artificial intelligence : A Modern Approach, Prentice Hall*
- *Artificial Intelligence by Elaine Rich & Kevin Knight*, Third Ed, Tata McGraw Hill
- *Artificial Intelligence and Expert System by Patterson*
- http://www.cs.rmit.edu.au/AI-Search/Product/
- http://aima.cs.berkeley.edu/demos.html (for more demos)
- Slides adapted from CS188 Instructor: Anca Dragan, University of California, Berkeley
- Slides adapted from CS60045 ARTIFICIAL INTELLIGENCE
- https://www.youtube.com/watch?v=huJEgJ82360



(some slides adapted from http://aima.cs.berkeley.edu/)