CSE 573: Artificial Intelligence
Autumn 2012

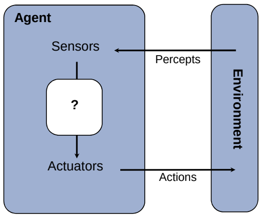
Introduction & Search

Dan Weld

With slides from
Dan Klein, Stuart Russell, Andrew Moore, Luke Zettlemoyer

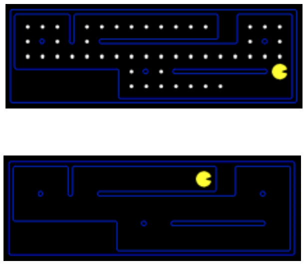
Agent *vs.* Environment

- An **agent** is an entity that *perceives* and *acts*.
- A **rational agent** selects actions that maximize its **utility function**.
- Characteristics of the **percepts, environment,** and **action space** dictate techniques for selecting rational actions.



Goal Based Agents

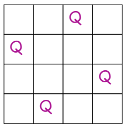
- Plan ahead
- Ask “what if”
- Decisions based on (hypothesized) consequences of actions
- Must have a model of how the world evolves in response to actions
- Act on how the world WOULD BE



Search thru a
Problem Space / State Space

- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state [test]
- Output:
  - Path: start ⇒ a state satisfying goal test
  - [May require shortest path]
  - [Sometimes just need state passing test]

Example: N Queens



- Input:
  - Set of states
  - Operators [and costs]
  - Start state
  - Goal state (test)
- Output

Machine Learning : predict fuel efficiency

Discrete Data

Predict MPG

mpg	cylinders	displacement	horsepower	weight	acceleration	modelyear	maker
good	4	low	low	low	high	75to78	asia
bad	6	medium	medium	medium	medium	70to74	america
bad	4	medium	medium	medium	low	75to78	europe
bad	8	high	high	high	low	70to74	america
bad	6	medium	medium	medium	medium	70to74	america
bad	4	low	medium	low	medium	70to74	asia
bad	4	low	medium	low	low	70to74	asia
bad	8	high	high	high	low	75to78	america
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:
bad	8	high	high	high	low	70to74	america
good	8	high	medium	high	high	79to83	america
bad	8	high	high	high	low	75to78	america
good	4	low	low	low	low	79to83	america
bad	6	medium	medium	medium	high	75to78	america
good	4	medium	low	low	low	79to83	america
good	4	low	low	medium	high	79to83	america
bad	8	high	high	high	low	70to74	america
good	4	low	medium	low	medium	75to78	europe
bad	5	medium	medium	medium	medium	75to78	europe

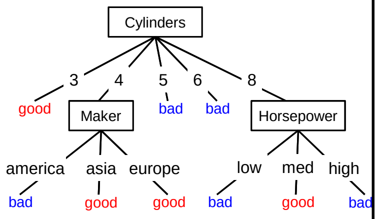
Y X

Need to find “Hypothesis”: $f : X \rightarrow Y$

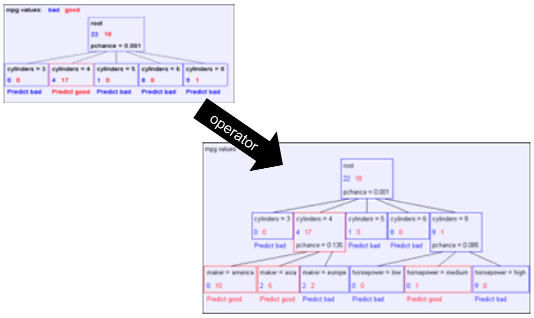
Hypotheses: decision trees $f : X \rightarrow Y$

- Each internal node tests an attribute x_i
- Each branch assigns an attribute value $x_i = v$
- Each leaf assigns a class y
- To classify input x?

  traverse the tree from root to leaf, output the labeled y



Search thru Space of Decision Trees

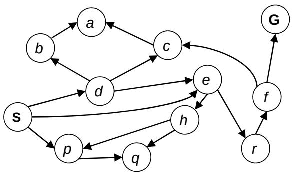


Search Methods

- Blind Search
  - Depth first search
  - Breadth first search
  - Iterative deepening search
  - Uniform cost search
- Local Search
- Informed Search
- Constraint Satisfaction
- Adversary Search

State Space Graphs

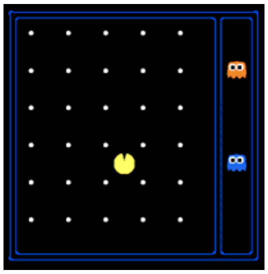
- State space graph:
  - Each node is a state
  - The successor function is represented by arcs
  - Edges may be labeled with costs
- We can rarely build this graph in memory (so we don't)



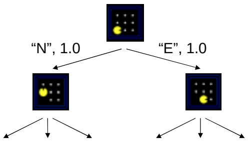
Ridiculously tiny search graph for a tiny search problem

State Space Sizes?

- Search Problem:
  Eat all of the food
- Pacman positions:
  10 x 12 = 120
- Pacman facing:
  up, down, left, right
- Food Count: 30
- Ghost positions: 12



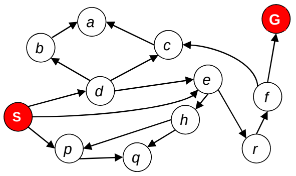
Search Trees



- A search tree:
  - Start state at the root node
  - Children correspond to successors
  - Nodes contain states, correspond to PLANS to those states
  - Edges are labeled with actions and costs
  - For most problems, we can never actually build the whole tree

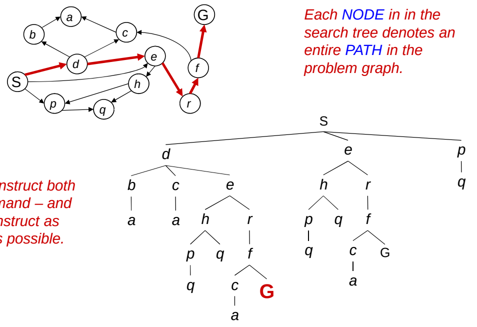
Example: Tree Search

State Graph:



What is the search tree?

State Graphs vs. Search Trees

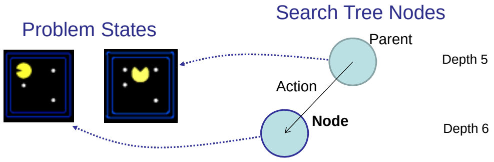


Each NODE in in the search tree denotes an entire PATH in the problem graph.

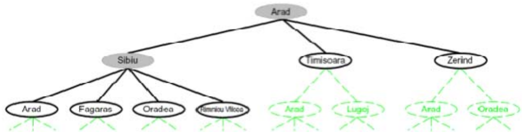
We construct both on demand – and we construct as little as possible.

States vs. Nodes

- Nodes in state space graphs are problem states
  - Represent an abstracted state of the world
  - Have successors, can be goal / non-goal, have multiple predecessors
- Nodes in search trees are plans
  - Represent a plan (sequence of actions) which results in the node's state
  - Have a problem state and one parent, a path length, a depth & a cost
  - The same problem state may be achieved by multiple search tree nodes

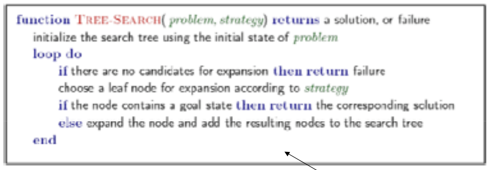


Building Search Trees



- Search:
  - Expand out possible plans
  - Maintain a fringe of unexpanded plans
  - Try to expand as few tree nodes as possible

General Tree Search



```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
  end
```

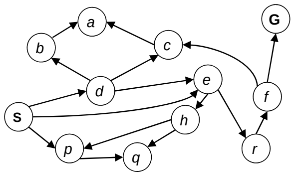
Detailed pseudocode is in the book!

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy
- Main question: which fringe nodes to explore?

Review: Depth First Search

***Strategy**: expand deepest node first*

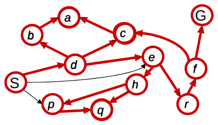
***Implementation**: Fringe is a LIFO queue (a stack)*

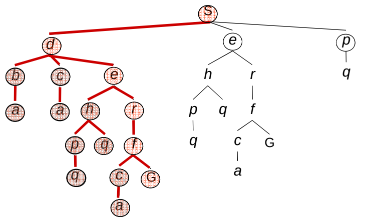


Review: Depth First Search

Expansion ordering:

(d,b,a,c,a,e,h,p,q,q,r,f,c,a,G)

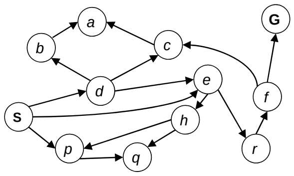




Review: Breadth First Search

Strategy: *expand shallowest node first*

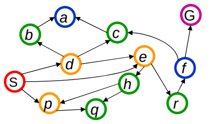
Implementation: *Fringe is a FIFO queue*

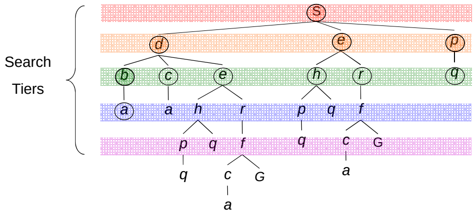


Review: Breadth First Search

Expansion order:

(S,d,e,p,b,c,e,h,r,q,a,a ,h,r,p,q,f,p,q,f,q,c,G)





Search Algorithm Properties

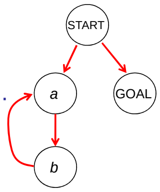
- Complete? Guaranteed to find a solution if one exists?
- Optimal? Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

Variables:

n	Number of states in the problem
b	The maximum branching factor *B* (the maximum number of successors for a state)
C^*	Cost of least cost solution
d	Depth of the shallowest solution
m	Max depth of the search tree

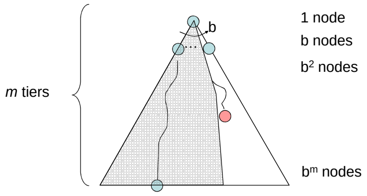
DFS

Algorithm		Complete	Optimal	Time	Space
DFS	Depth First Search	No	No	Infinite	Infinite



- Infinite paths make DFS incomplete…
  - How can we fix this?
  - Check new nodes against path from S
- Infinite search spaces still a problem

DFS

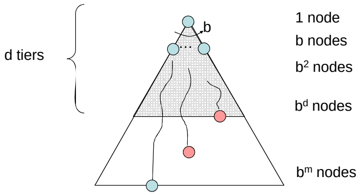


Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y if finite	N	$O(b^m)$	$O(bm)$

* Or graph search – next lecture.

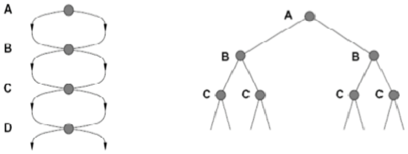
BFS

Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	Y	$O(b^d)$	$O(b^d)$



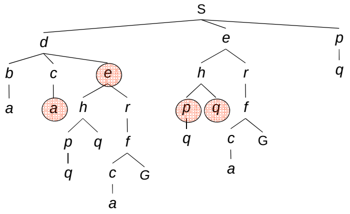
Extra Work?

- Failure to detect repeated states can cause exponentially more work (why?)



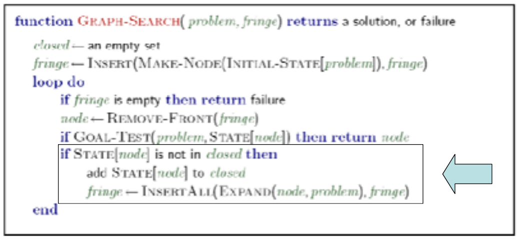
Graph Search

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



Graph Search

- Very simple fix: never expand a state type twice



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

- Can this wreck completeness? Why or why not?
- How about optimality? Why or why not?

Some Hints

- Graph search is almost always better than tree search (when not?)
- Implement your closed list as a dict or set!
- Nodes are conceptually paths, but better to represent with a state, cost, last action, and reference to the parent node

Memory a Limitation?

- Suppose:
  - 4 GHz CPU
  - 6 GB main memory
  - 100 instructions / expansion
  - 5 bytes / node
- 400,000 expansions / sec
  - Memory filled in 300 sec … 5 min

Comparisons

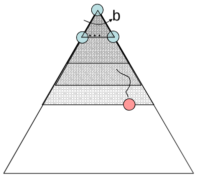
- When will BFS outperform DFS?
- When will DFS outperform BFS?

Iterative Deepening

Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less.
2. If “1” failed, do a DFS which only searches paths of length 2 or less.
3. If “2” failed, do a DFS which only searches paths of length 3 or less.

….and so on.



Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	Y	$O(b^d)$	$O(b^d)$
ID		Y	Y	$O(b^d)$	$O(bd)$

Cost of Iterative Deepening

b	ratio ID to DFS
2	3
3	2
5	1.5
10	1.2
25	1.08
100	1.02

Speed

Assuming 10M nodes/sec & sufficient memory

	BFS			Iter. Deep.	
	Nodes	Time		Nodes	Time
8 Puzzle	10^5	.01 sec		10^5	.01 sec
2x2x2 Rubik’s	10^6	.2 sec		10^6	.2 sec
15 Puzzle	10^{13}	6 days	1Mx	10^{17}	20k yrs
3x3x3 Rubik’s	10^{19}	68k yrs	8x	10^{20}	574k yrs
24 Puzzle	10^{25}	12B yrs		10^{37}	10^{23} yrs

Why the difference?

Rubik has higher branching factor

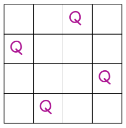
15 puzzle has greater depth

# of duplicates

Slide adapted from Richard Korf presentation

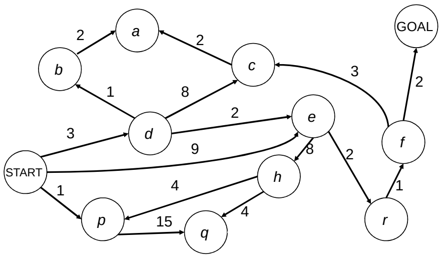
When to Use Iterative Deepening

- N Queens?



© Daniel S. Weld

Costs on Actions

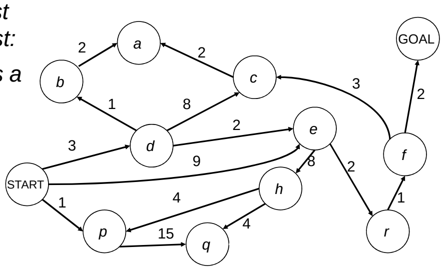


Notice that BFS finds the shortest path in terms of number of transitions. It does not find the least-cost path.

Uniform Cost Search

Expand cheapest node first:

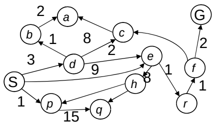
*Fringe is a **priority queue***

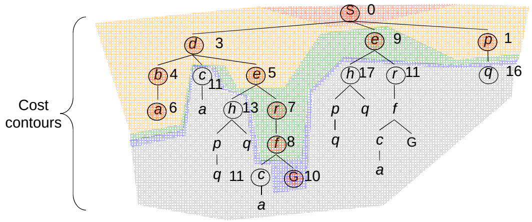


Uniform Cost Search

Expansion order:

(S,p,d,b,e,a,r,f,e,G)





Priority Queue Refresher

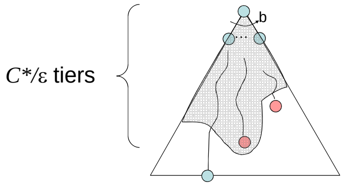
- A priority queue is a data structure in which you can insert and retrieve (key, value) pairs with the following operations:

pq.push(key, value)	inserts *(key, value)* into the queue.
pq.pop()	returns the key with the lowest value, and removes it from the queue.

- You can decrease a key's priority by pushing it again
- Unlike a regular queue, insertions aren't constant time, usually O(log *n*)
- We'll need priority queues for cost-sensitive search methods

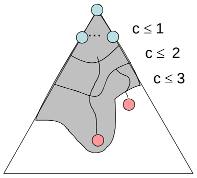
Uniform Cost Search

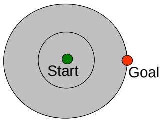
Algorithm		Complete	Optimal	Time	Space
DFS	w/ Path Checking	Y	N	$O(b^m)$	$O(bm)$
BFS		Y	Y	$O(b^d)$	$O(b^d)$
UCS		Y*	Y	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$



Uniform Cost Issues

- Remember: explores increasing cost contours
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location





Uniform Cost: Pac-Man

- Cost of 1 for each action
- Explores all of the states, but one

Exponentials Everywhere

"I think we're going to need a stronger donkey…"

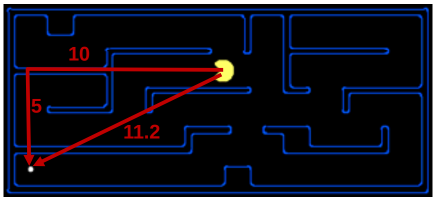
Heuristics

RULE OF THUMB

If your extended thumb is too small to block your view of the hazmat incident, you're not far enough away.

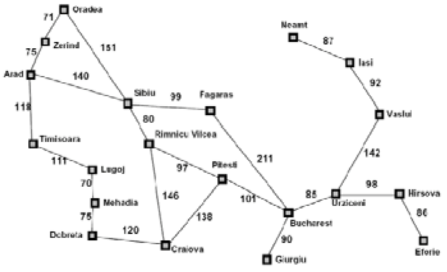
Search Heuristics

- Any *estimate* of how close a state is to a goal
- Designed for a particular search problem



- Examples: Manhattan distance, Euclidean distance

Heuristics



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

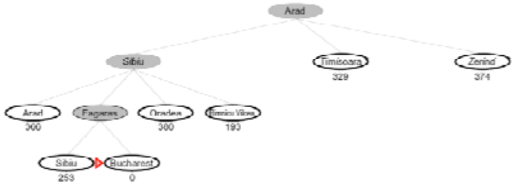
Best First / Greedy Search

Expand closest node first: Fringe is a priority queue

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

Best First / Greedy Search

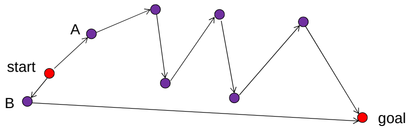
- Expand the node that seems closest…



- What can go wrong?

Greedy Search

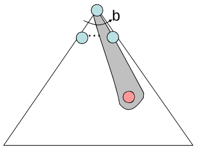
Expand the node that seems closest…

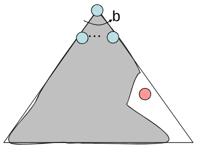


What can go wrong?

Best First / Greedy Search

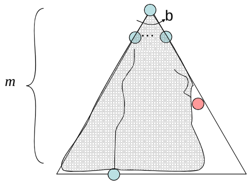
- A common case:
  - Best-first takes you straight to the (wrong) goal
- Worst-case: like a badly-guided DFS in the worst case
  - Can explore everything
  - Can get stuck in loops if no cycle checking
- Like DFS in completeness (finite states w/ cycle checking)





Best First Greedy Search

Algorithm	Complete	Optimal	Time	Space
Greedy Best-First Search	Y*	N	$O(b^m)$	$O(b^m)$



- What do we need to do to make it complete?
- Can we make it optimal?

A* Search

Hart, Nilsson & Rafael 1968

Best first search with $f(n) = g(n) + h(n)$

- $g(n)$ = sum of costs from start to n
- $h(n)$ = estimate of lowest cost path n → goal

  $h(goal) = 0$

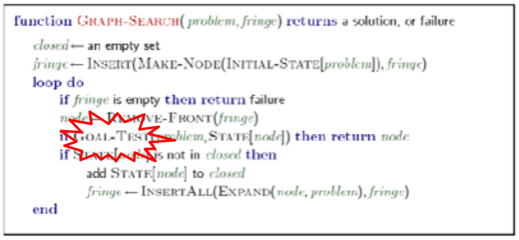
If h(n) is admissible and monotonic

then A* is optimal

Underestimates cost of reaching goal from node

f values increase from node to descendants (triangle inequality)

Graph Search Detail

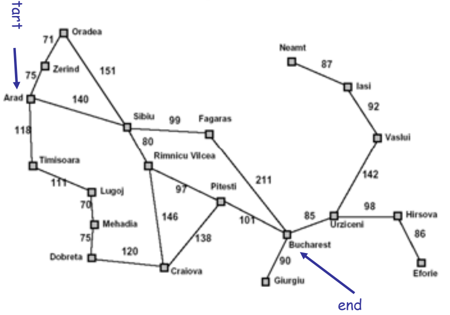


```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node])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

When do we check for goals?

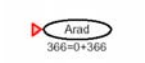
- ~~When adding to queue?~~
- When removing from queue?

European Example

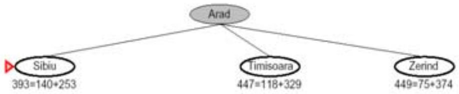


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

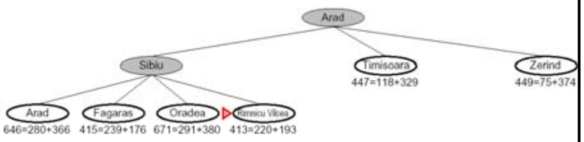
A* Example



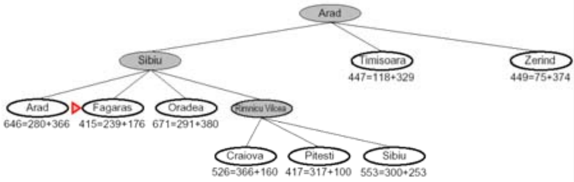
A* Example



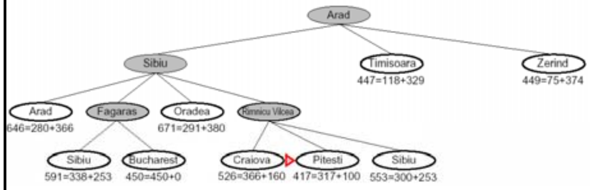
A* Example



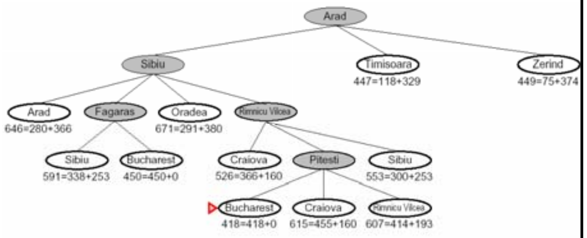
A* Example



A* Example

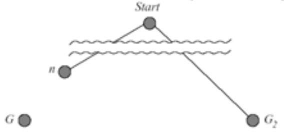


A* Example



Optimality of A*

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



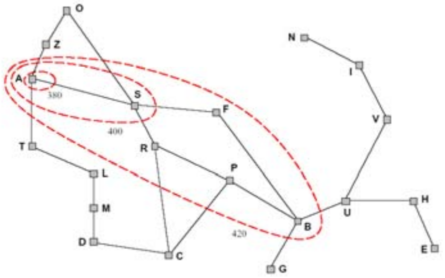
$$\begin{aligned} f(G_2) &= g(G_2) && \text{since } h(G_2)=0 \\ &> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\ &\geq f(n) && \text{since } h \text{ is admissible} \end{aligned}$$

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

Optimality Continued

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

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

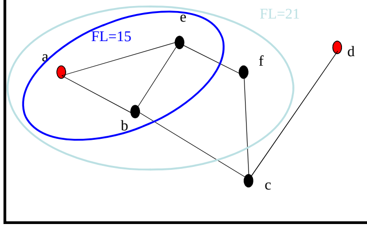


A* Summary

- Pros
- Cons

Iterative-Deepening A*

- Like iterative-deepening depth-first, but...
- Depth bound modified to be an f-limit
  - Start with f-limit = h(start)
  - Prune any node if f(node) > f-limit
  - Next f-limit = min-cost of any node pruned



IDA* Analysis

- Complete & Optimal (ala A*)
- Space usage ∝ depth of solution
- Each iteration is DFS - no priority queue!
- # nodes expanded relative to A*
  - Depends on # unique values of heuristic function
  - In 8 puzzle: few values ⇒ close to # A* expands
  - In traveling salesman: each f value often unique

    ⇒ 1+2+…+n = O(n^2) where n=nodes A* expands

    if n is too big for main memory, n^2 is too long to wait!
- Generates duplicate nodes in cyclic graphs

Forgetfulness

- A* used exponential memory
- How much does IDA* use?
  - During a run?
  - In between runs?

SMA*

- Use all available memory
- Start like A*
- When memory is full…
    - Erase node with highest f-value
    - First, backup parent with this f-value
    - So… parent knows cost-bound on best child

Alternative Approach to Finite Memory…

- Optimality is nice to have, but…

Depth-First Branch & Bound

- Single DF search
    - → uses linear space
- Keep track of best solution so far
- If $f(n) = g(n)+h(n) \geq cost(best\text{-}soln)$
    - Then prune n
- Requires
    - Finite search tree, or
    - Good upper bound on solution cost
- Generates duplicate nodes in cyclic graphs

Tradeoff:
Prune space, but…
Must apply test to each node

Adapted from Richard Korf presentation

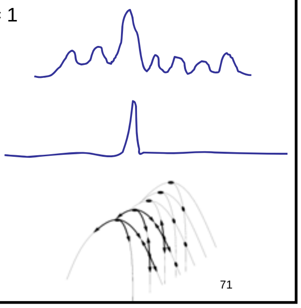
Beam Search

- Idea
    - Best first but only keep N best items on priority queue
- Evaluation
    - Complete?
    - Time Complexity?
    - Space Complexity?

Hill Climbing

"Gradient ascent"

- Idea
    - Always choose best child; no backtracking
    - Beam search with |queue| = 1
- Problems?
    - Local maxima
    - Plateaus
    - Diagonal ridges

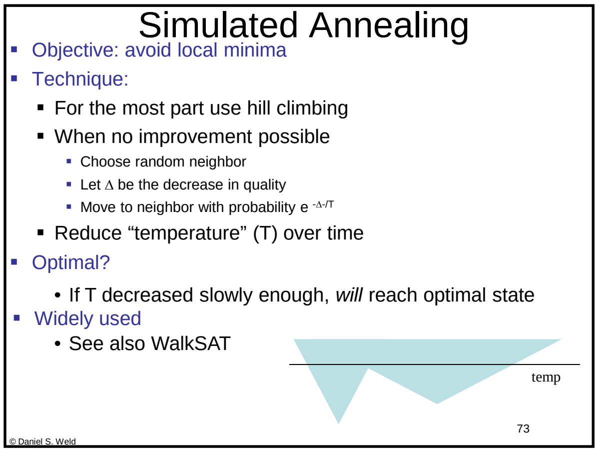


Randomizing Hill Climbing

- Randomly disobeying heuristic
- Random restarts

(heavy tailed distributions)

→ Local Search



Simulated Annealing

- Objective: avoid local minima
- Technique:
  - For the most part use hill climbing
  - When no improvement possible
    - Choose random neighbor
    - Let Δ be the decrease in quality
    - Move to neighbor with probability e $^{-\Delta-/T}$
  - Reduce “temperature” (T) over time
- Optimal?
  - If T decreased slowly enough, *will* reach optimal state
- Widely used
  - See also WalkSAT

© Daniel S. Weld