Start reading Ch. 7, 8, 9

Project topics

Course web site

http://www.cs.illinois.edu/class/cs440/

or

http://www.cs.uiuc.edu/class/fa10/cs440

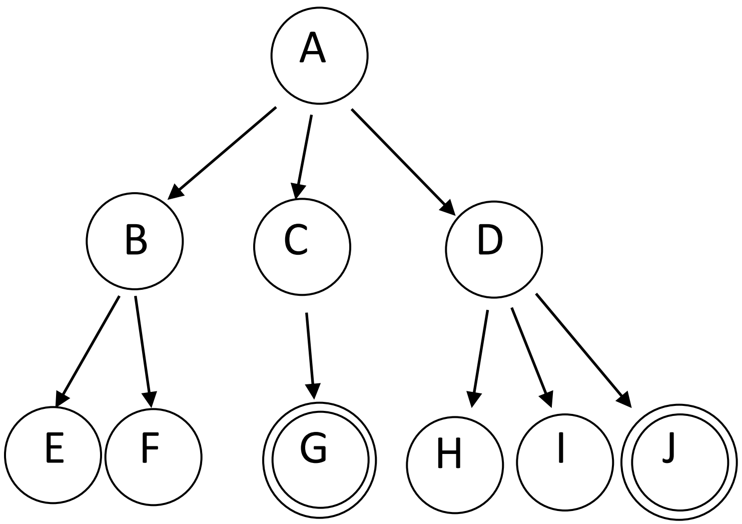
Generic Search Function

```
SEARCH(Problem P, Queuing Function QF):
  local:    n    /* current node */
            q    /* nodes to explore */
  q ← singleton of Initial_State(P);
  Loop:
    if q = () return failure;
    n ← Pop(q);
    if n Solves P return n;
    q ← QF(q, Expand(n));
  end
```

Depth First: `QF(old, new): Append(new, old);`

Breadth First: `QF(old, new): Append(old, new);`

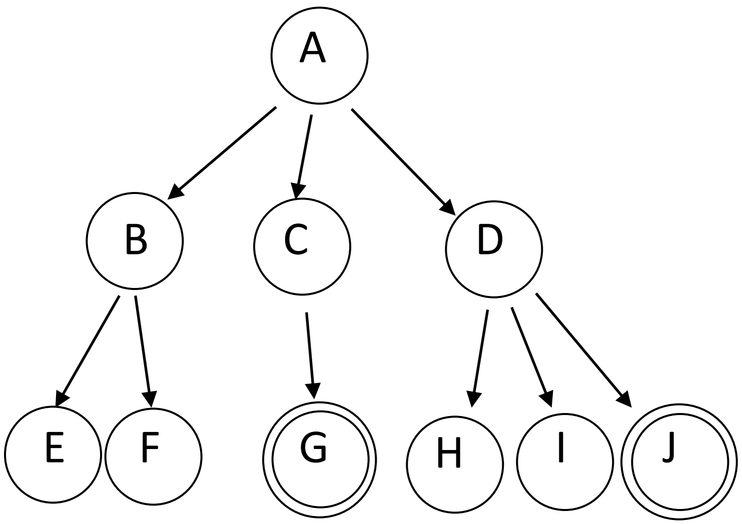
Sample Search Tree



Depth First

n	q (between iterations)
-	(A)
A	(B C D)
B	(E F C D)
E	(F C D)
F	(C D)
C	(G D)
G	(D)

Sample Search Tree



Breadth First

n	q
-	(A)
A	(B C D)
B	(C D E F)
C	(D E F G)
D	(E F G H I J)
E	(F G H I J)

• • •

Costs

- Performing the search to find Goal
  - Time
  - Space
- Executing an operator in the world
- We will focus on execution cost
- We assume
  - Positive finite cost for each action
  - Finite branching factor
- Important cost functions: g, h, f

Define: $g^*(n)$ as minimum cost from root to n $\forall n \in$ Nodes

Define: $g(n)$ as an easily computable approximation to g^*

How might we compute g?

Need an execution cost model for each operator

How might $g(n) \neq g^*(n)$?

In fact it is usually easy to guarantee $g(n) = g^*(n)$
(especially for trees!)

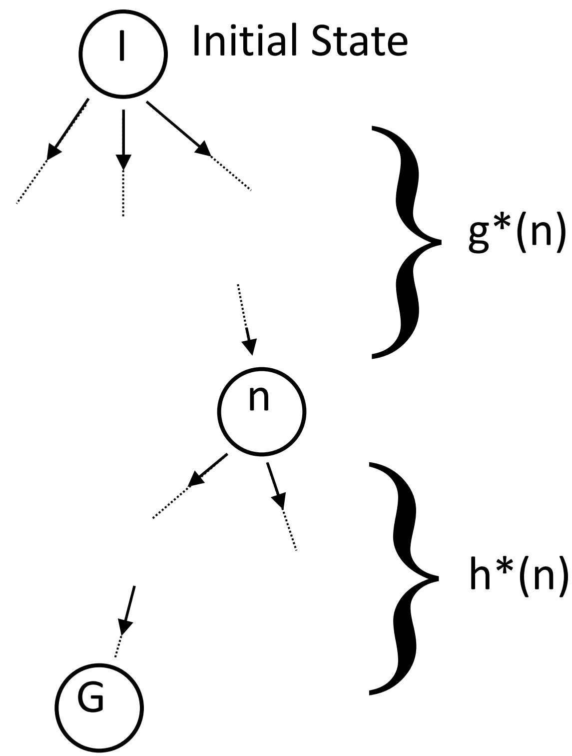
Note the node alone determines which operators can apply next (First Order Markov: history is unimportant to world dynamics
– more later)

Define: $h^*(n)$ as minimum cost from n to a goal $\forall n \in Nodes$

Define: $h(n)$ as an easily computable approximation to h^*

h is called a "heuristic function"

Define: $f(n)$ as $g(n) + h(n)$



Lowest cost solution constrained through n

f(n) approximates this

A search algorithm (together with its heuristic function if needed) is *admissible* iff for all search trees:

A) If there exists a goal, the search will not fail.

B) If there are multiple goals the search will find the best (least expensive to execute) goal.

Search Function

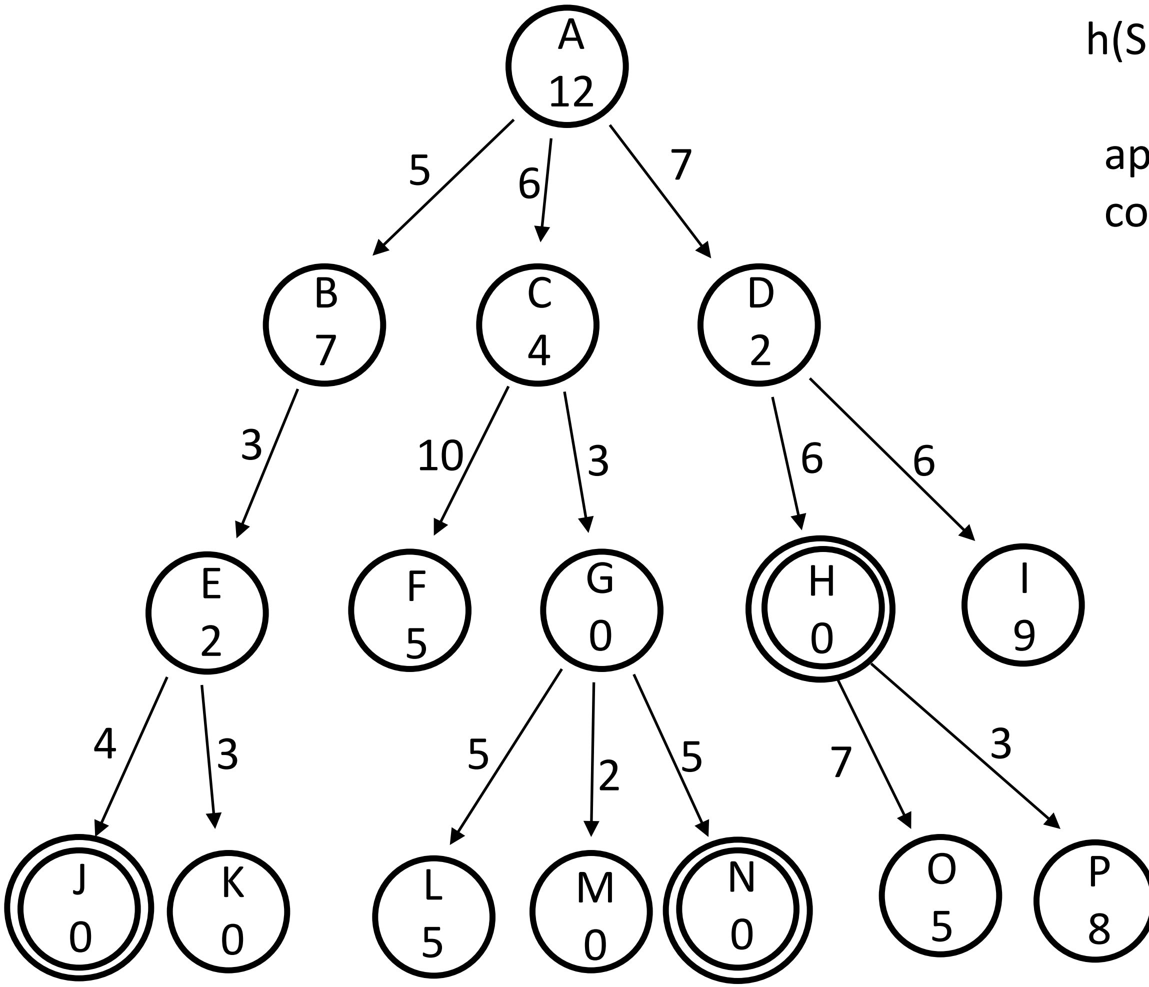
```
SEARCH(Problem P, Queuing Function QF):
  local:    n    /* current node */
            q    /* nodes to explore */
  q ← singleton of Initial_State(P);
  Loop:
    if q = () return failure;
    n ← Pop(q);
    if n Solves P return n;
    q ← QF(q, Expand(n));
  end

QF(a,b): Sort(Append(a,b), [ ])

  [ ]=
```

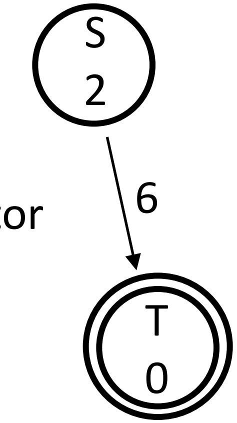
g → uniform cost
h → best first / greedy
f → A / A*

Complete Search Tree



h(S)=2

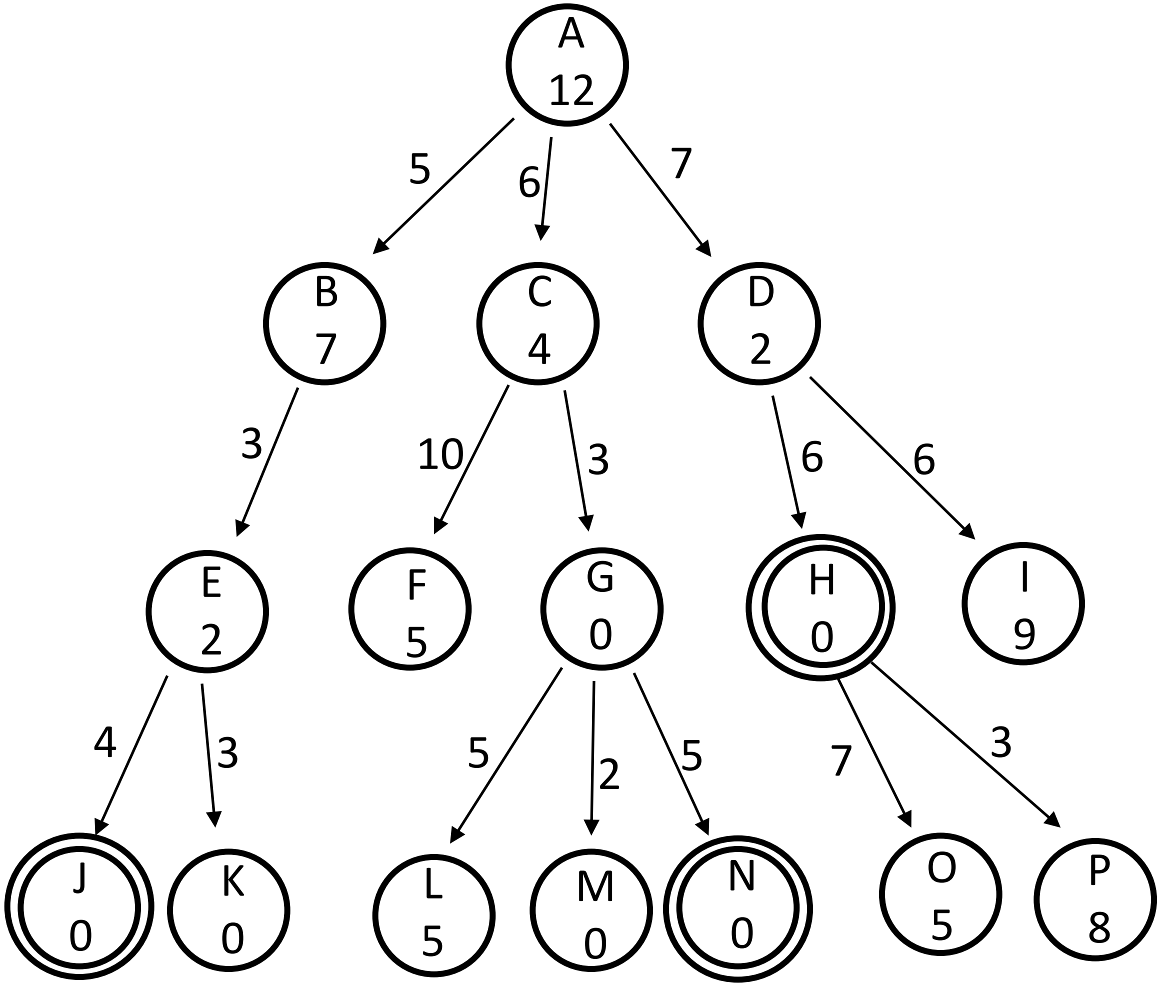
application of operator costs 6



double circle indicates a goal

Remember the notation. We'll use it throughout our search discussion

Complete Search Tree



Uniform Cost

n	**q**
-	**(A)**
A	**(B C D)**
B	**(C D E)**
C	**(D E G F)**
D	**(E G H I F)**
E	**(G K J H I F)**
G	**(M K J H I L N F)**
M	**(K J H I L N F)**
K	**(J H I L N F)**
J	**(H I L N F)**

local variables n and q between iterations of loop block

Best First / Greedy

n	q
-	(A)
A	(D C B)
D	(H C B I)
H	(C B I)

Uniform Cost

n	q
-	(A)
A	(B C D)
B	(C D E)
C	(D E G F)
D	(E G H I F)
E	(G K J H I F)
G	(M K J H I L N F)
M	(K J H I L N F)
K	(J H I L N F)
J	(H I L N F)

A / A*

n	q
-	(A)
A	(D C B)
D	(C B H I)
C	(G B H F I)
G	(M B H N L F I)
M	(B H N L F I)
B	(E H N L F I)
E	(K J H N L F I)
K	(J H N L F I)
J	(H N L F I)

Example: 8-Puzzle

4	8	2
1	6	
5	3	7

Initial State

1	2	3
4	5	6
7	8	

Goal

Four Operators:

MoveTileUp
MoveTileDown
MoveTileLeft
MoveTileRight

(preconditions and effects)
(alternative possibilities)

Aside: What is a Solution?

- What is the 8-puzzle goal?
- Sometimes the path *is* the goal…
- What about cryptarithmetic?
- What individuates a Node?
- Searching graphs...

Possible Heuristic Functions

Often simplify by relaxing some constraint

- h_1: 3
- h_2: count tiles out of place
- h_3: sum Manhattan metric distances

Admissibility of A*

Some authors use “A” if not met

1) $\forall n \; \forall n' \in$ nodes, $\forall o \in$ operators with $o(n) \rightarrow n'$

$$h(n) \leq cost(n,o,n') + h(n')$$

Triangle inequality, Montonicity, or Consistency

(for trees…)

Admissibility of A* (cont)

2) $\forall n \in$ nodes

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

Informally: be optimistic
(or don't be pessimistic)
Why? Could you prove it?

Important General Principle:
Optimism Under Uncertainty

Does not depend on problem or tree!

Is "Uniform Cost" admissible?

We have:

A_1* with heuristic fcn h_1

A_2* with heuristic fcn h_2

A_1* and A_2* are admissible

Then we say

A_1* is *more informed* than A_2*

iff for all non-goal nodes n

$h_1(n) > h_2(n)$

"more informed" implies "guaranteed not to search more"

- Achievement or Satisfiability goals
  - Goal is achieved or it is not
  - Later: zero / one loss function
  - Combinatorial
- Optimization goals
  - Do the best you can
  - Continuous loss
  - Local / efficient: Gradient information
  - “Convex optimization” is influential today

Different Search Protocols

- Satisfiability - solve the problem
  - previous searches
- Optimizing - maximize utility
  - alterations needed
- Satisficing - good enough optimization
  - Combination; somewhat informal

Optimizing / Satisficing require some kind of metric space

$$U: N \rightarrow \Re$$

Utility function maps nodes to real numbers; higher is better

Locally Optimizing Search

- States have utilities
- Smoothness properties
- Want highest utility node
- Heuristic fcn can be used to compute utility as 1/h or - h
  - larger utility (smaller h) is better
  - SORT still orders nodes <u>best</u> to <u>worst</u>
- Some access to the local gradient

Hill Climbing, Optimizing Beam

(steepest *descent* if directly using h as before)

Search Function

```
SEARCH(Problem P, Queuing Function QF):
  local:    n    /* current node */
            q    /* nodes to explore */
  q ← singleton of Initial_State(P);
  Loop:
    if q = () return failure;
    n ← Pop(q);
    if n Solves P return n;
    q ← QF(q, Expand(n));
  end

QF(a,b): Sort(Append(a,b), [ ])

  [ ]=
```

g - uniform cost
h - best first / greedy
f - A / A*

Locally Optimizing Search: Hill Climbing

```
SEARCH(Problem P, Queuing Function QF):
  local:    n    /* current node */
            q    /* nodes to explore */
  q ← singleton of Initial_State(P);
  Loop:
    if q = () return n;
    if First(q) < n return n;
    n ← Pop(q);
    q ← QF(q, Expand(n));
  end

QF(a,b): Sort(Append(a,b), h);
           Delete all but best;
```

Hill Climbing Difficulties

- Foothill - local maximum
- Plateau - little information
- Ridge - can’t go that way

An Aside

(but an important conceptual generalization)

- Note the efficiency of using local guidance
  - Metric
  - Smooth
  - Gradient
- Conditions for global solution?
  - No foothills, no plateaus, no ridges…
  - Local (linearized) information
    - Qualitatively correct
    - Although quantitatively incorrect
- Convex Optimization

Recall Best First / Greedy

```
SEARCH(Problem P, Queuing Function QF):
  local:    n    /* current node */
            q    /* nodes to explore */
  q ← singleton of Initial_State(P);
  Loop:
    if q = () return failure;
    n ← Pop(q);
    if n Solves P return n;
    q ← QF(q, Expand(n));
  end

QF(a,b): Sort(Append(a,b), h);
```

Beam Search w/ beam width k

```
SEARCH(Problem P, Queuing Function QF):
  local:    n    /* current node */
            q    /* nodes to explore */
  q ← singleton of Initial_State(P);
  Loop:
    if q = () return failure;
    n ← Pop(q);
    if n Solves P return n;
    q ← QF(q, Expand(n));
  end

QF(a,b): Sort(Append(a,b), h);
           Delete all but the best k;
```

Locally Optimizing Beam Search

w/ beam width k

- Variable “n” becomes a vector of k nodes
- Find all descendants at once
- Sort descendant list by “h” and delete all but the best k
- Return as in Hill Climbing using best(n)

Search Properties

- Tentative (don't throw away information)

  Depth First | Breadth First | Uniform Cost
  Best First/Greedy | A*

- Irrevocable (throw away information)

  Hill Climbing | Beam | Optimizing Beam

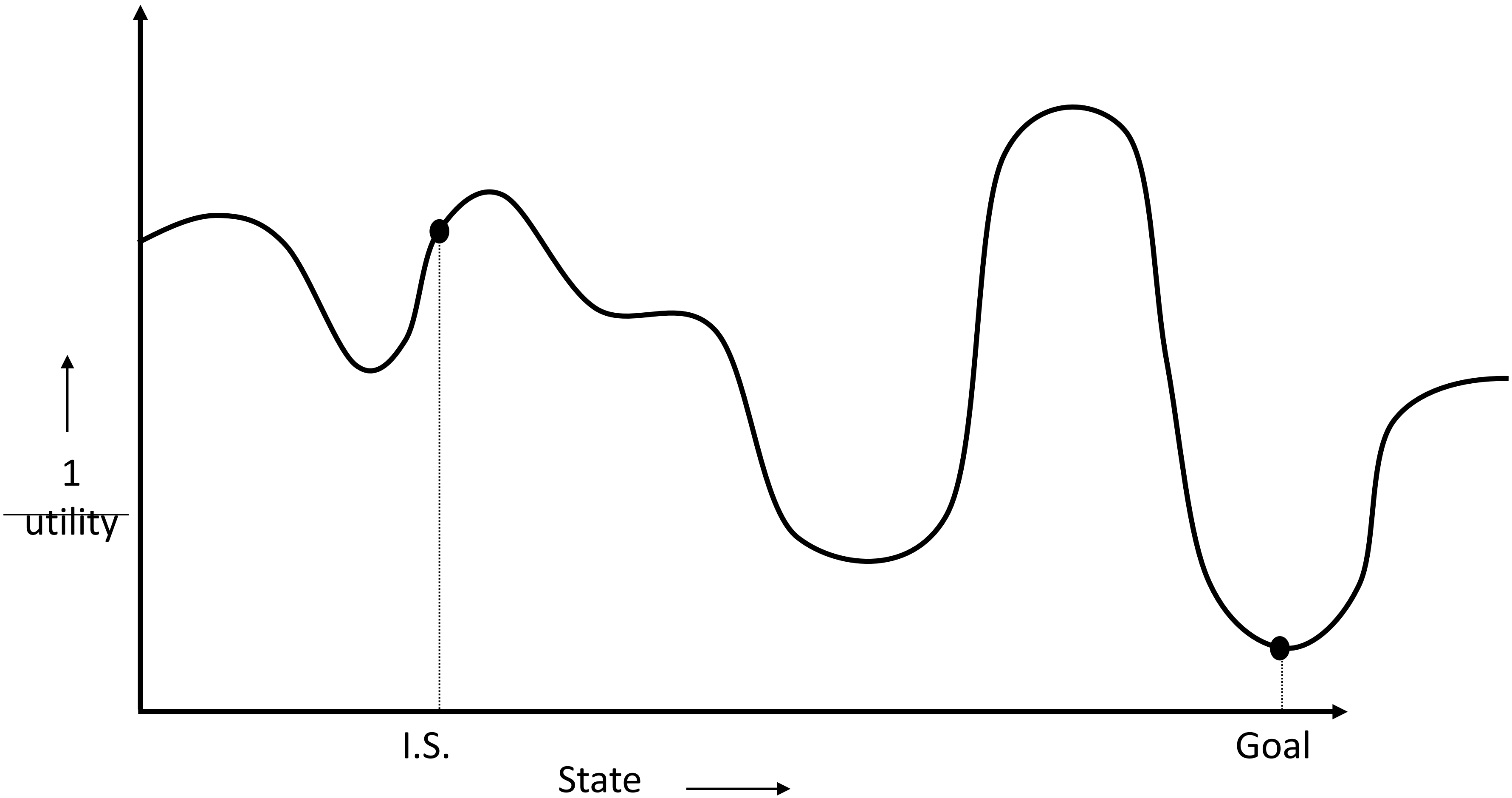
- Exhaustive (will visit all nodes or find goal)

  Breadth First | Uniform Cost | A*

- Admissible

  Uniform Cost | A* [if consistency & optimism]

Non-Systematic Search: Simulated Annealing



Simulated Annealing

- Analogy w/ metalurgy
- Add thermal energy; noise; randomness
- Propose / accept actions probabilistically
- Initially noise / randomness dominates
- Cooling schedule: converge to determinism
- Guarantees?
- Compare random restart hill-climbing