Artificial Intelligence

Informed Search

Chapter 4

Adapted from materials by Tim Finin,
Marie desJardins, and Charles R. Dyer

Outline

- Heuristic search
- Best-first search
  - Greedy search
  - Beam search
  - A, A*
  - Examples
- Memory-conserving variations of A*
- Heuristic functions
- Iterative improvement methods
  - Hill climbing
  - Simulated annealing
  - Local beam search
  - Genetic algorithms
- Online search

Heuristic

Merriam-Webster's Online Dictionary

Heuristic (pron. \hy*u*-ˈris-tik\): adj. [from Greek *heuriskein* to discover.] involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods

The Free On-line Dictionary of Computing (15Feb98)

heuristic 1. <programming> A rule of thumb, simplification or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood. Unlike algorithms, heuristics do not guarantee feasible solutions and are often used with no theoretical guarantee. 2. <algorithm> approximation algorithm.

From WordNet (r) 1.6

heuristic adj 1: (computer science) relating to or using a heuristic rule 2: of or relating to a general formulation that serves to guide investigation [ant: algorithmic] n : a commonsense rule (or set of rules) intended to increase the probability of solving some problem [syn: heuristic rule, heuristic program]

Informed methods add domain-specific information

- Add domain-specific information to select the best path along which to continue searching
- Define a heuristic function ***h*(*n*)** that estimates the "goodness" of a node *n*.
  - Specifically, $h(n)$ = **estimated cost** (or distance) of minimal cost path from *n* **to a goal state**.
- The heuristic function is an estimate of how close we are to a goal, based on domain-specific information that is computable from the current state description.

Heuristics

- **All domain knowledge** used in the search is encoded in the **heuristic function** ***h*()**.
- Heuristic search is an example of a "**weak method**" because of the limited way that domain-specific information is used to solve the problem.
- Examples:
  - Missionaries and Cannibals: Number of people on starting river bank
  - 8-puzzle: Number of tiles out of place
  - 8-puzzle: Sum of distances each tile is from its goal position
- In general:
  - $h(n) \geq 0$ for all nodes n
  - $h(n) = 0$ implies that n is a goal node
  - $h(n) = \infty$ implies that n is a dead-end that can never lead to a goal

Weak vs. strong methods

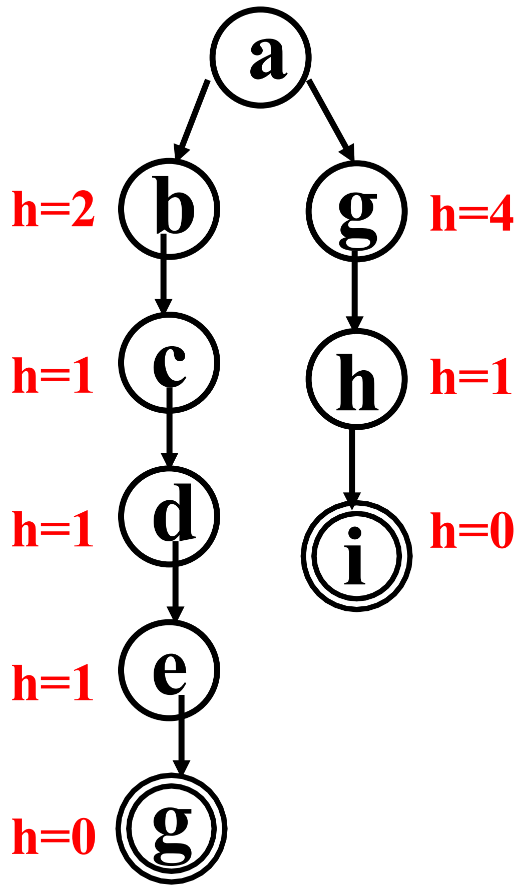
- We use the term *weak methods* to refer to methods that are extremely *general* and not tailored to a specific situation.
- Examples of weak methods include
  - **Means-ends analysis** is a strategy in which we try to represent the current situation and where we want to end up and then look for ways to shrink the differences between the two.
  - **Space splitting** is a strategy in which we try to list the possible solutions to a problem and then try to rule out classes of these possibilities.
  - **Subgoaling** means to split a large problem into several smaller ones that can be solved one at a time.
- Called "weak" methods because they do not take advantage of more powerful domain-specific heuristics

Best-first search

- Order nodes on the nodes list by increasing value of an evaluation function ***f*** **(*n*)**
  - $f(n)$ incorporates domain-specific information in some way.
- This is a generic way of referring to the class of informed methods.
  - We get different searches depending on the evaluation function $f(n)$

Greedy search

- Use as an evaluation function $f(n) = h(n)$, sorting nodes by increasing values of f.
- Selects node to expand believed to be **closest** (hence "greedy") to a goal node (i.e., select node with smallest f value)
- Not complete
- Not admissible, as in the example.
  - Assuming all arc costs are 1, then greedy search will find goal g, which has a solution cost of 5.
  - However, the optimal solution is the path to goal I with cost 3.



Beam search

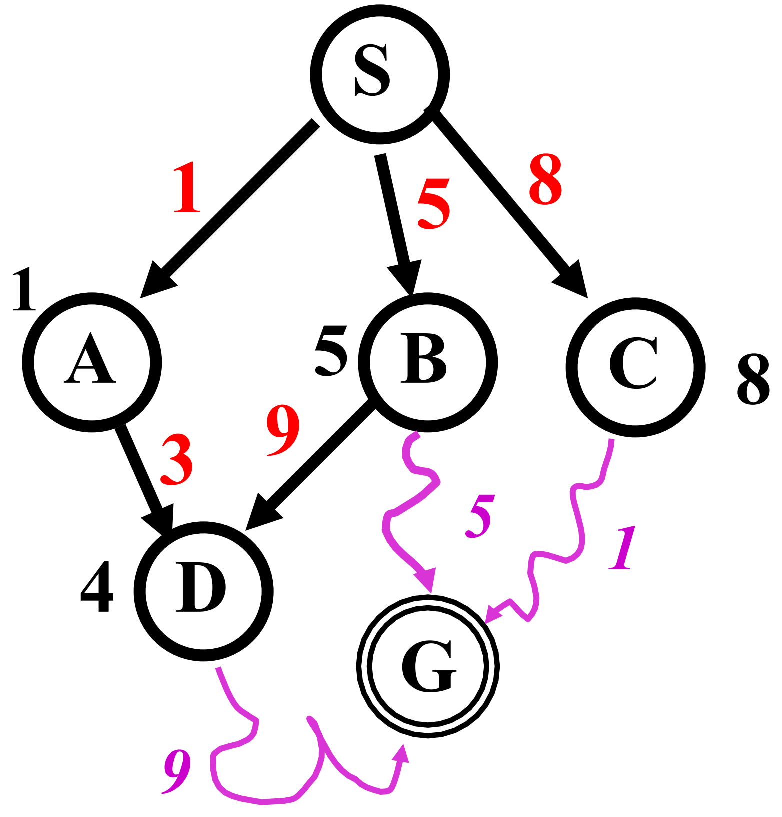
- Use an evaluation function $f(n) = h(n)$, but the maximum size of the nodes list is k, a fixed constant
- Only keeps k best nodes as candidates for expansion, and throws the rest away
- More space efficient than greedy search, but may throw away a node that is on a solution path
- Not complete
- Not admissible

Algorithm A

- Use as an evaluation function

  $\boldsymbol{f(n) = g(n) + h(n)}$

- $g(n)$ = minimal-cost path from the start state to state n.
- The $g(n)$ term adds a “breadth-first” component to the evaluation function.
- Ranks nodes on search frontier by estimated cost of solution from start node through the given node to goal.
- Not complete if $h(n)$ can equal infinity.
- Not admissible.



g(d)=4

h(d)=9

C is chosen next to expand

Algorithm A

1. Put the start node S on the nodes list, called OPEN
2. If OPEN is empty, exit with failure
3. Select node in OPEN with minimal $f(n)$ and place on CLOSED
4. If n is a goal node, collect path back to start and stop.
5. Expand n, generating all its successors and attach to them pointers back to n. For each successor n' of n
    1. If n' is not already on OPEN or CLOSED
        - put n' on OPEN
        - compute $h(n')$, $g(n') = g(n) + c(n,n')$, $f(n') = g(n') + h(n')$
    2. If n' is already on OPEN or CLOSED and if $g(n')$ is lower for the new version of n', then:
        - Redirect pointers backward from n' along path yielding lower $g(n')$.
        - Put n' on OPEN.

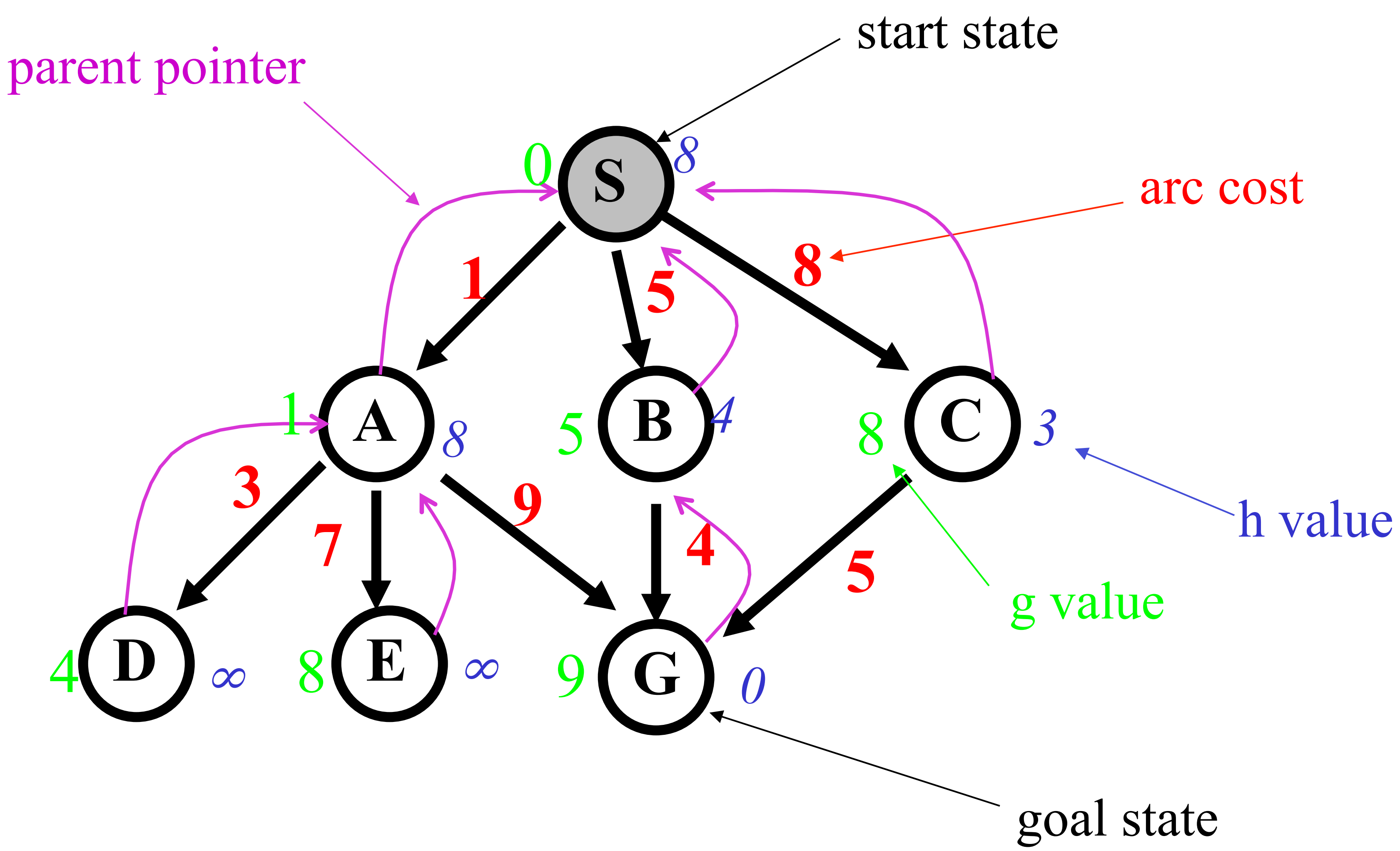
Algorithm A*

- Algorithm A with constraint that $\boldsymbol{h(n) \leq h^*(n)}$
  - $\boldsymbol{h^*(n)}$ = true cost of the minimal cost path from n to a goal.
- Therefore, $\boldsymbol{h(n)}$ is an **underestimate** of the distance to the goal.
- $h()$ is **admissible** when $h(n) \leq h^*(n)$ holds.
- Using an admissible heuristic guarantees that the first solution found will be an optimal one.
- A* is **complete** whenever the branching factor is finite, and every operator has a fixed positive cost
- A* is **admissible**

Some observations on A

- **Perfect heuristic:** If $h(n) = h^*(n)$ for all n, then only the nodes on the optimal solution path will be expanded. So, no extra work will be performed.
- **Null heuristic:** If $h(n) = 0$ for all n, then this is an admissible heuristic and A* acts like Uniform-Cost Search.
- **Better heuristic:** If $h_1(n) < h_2(n) \leq h^*(n)$ for all non-goal nodes, then h_2 is a better heuristic than h_1
  - If A_1* uses h_1, and A_2* uses h_2, then every node expanded by A_2* is also expanded by A_1*.
  - In other words, A_1 expands at least as many nodes as A_2*.
  - We say that A_2* is better informed than A_1*.
- The closer h is to h^*, the fewer extra nodes that will be expanded

Example search space



In-class Example

n	$\boldsymbol{g(n)}$	$\boldsymbol{h(n)}$	$\boldsymbol{f(n)}$	$\boldsymbol{h^*(n)}$
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	inf	inf	inf
E	8	inf	inf	inf
G	9	0	9	0

- $h^*(n)$ is the (hypothetical) perfect heuristic.
- Since $h(n) \leq h^*(n)$ for all n, h is admissible
- Optimal path = S B G with cost 9.

Greedy search

$f(n) = h(n)$

```
node expanded      nodes list
                 { S(8) }
     S           { C(3) B(4) A(8) }
     C           { G(0) B(4) A(8) }
     G           { B(4) A(8) }
```

- Solution path found is S C G, 3 nodes expanded.
- Wow, that is a fast search!! But it is NOT optimal.

A* search

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

```
node exp.      nodes list
            { S(8) }
 S          { A(9) B(9) C(11) }
 A          { B(9) G(10) C(11) D(inf) E(inf) }
 B          { G(9) G(10) C(11) D(inf) E(inf) }
 G          { C(11) D(inf) E(inf) }
```

- Solution path found is S B G, 4 nodes expanded..
- Still pretty fast. And optimal, too.

Proof of the optimality of A*

- We assume that A* has selected G2, a goal state with a suboptimal solution ($g(G2) > f^*$).
- We show that this is impossible.
  - Choose a node n on the optimal path to G.
  - Because $h(n)$ is admissible, $f(n) \leq f^*$.
  - If we choose G2 instead of n for expansion, $f(G2) \leq f(n)$.
  - This implies $f(G2) \leq f^*$.
  - G2 is a goal state: $h(G2) = 0$, $f(G2) = g(G2)$.
  - Therefore $g(G2) \leq f^*$
  - Contradiction.

Dealing with hard problems

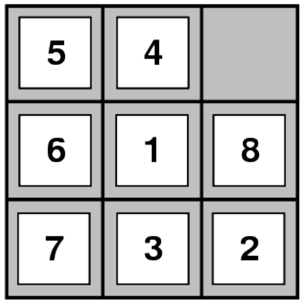
- For large problems, A* often requires too much space.
- Two variations conserve memory: IDA* and SMA*
- IDA* -- iterative deepening A*
  - uses successive iteration with growing limits on *f*. For example,
    - A* but don't consider any node *n* where *f* (*n*) > 10
    - A* but don't consider any node *n* where *f* (*n*) > 20
    - A* but don't consider any node *n* where *f* (*n*) > 30, ...
- SMA* -- Simplified Memory-Bounded A*
  - uses a queue of restricted size to limit memory use.
  - throws away the "oldest" worst solution.

What's a good heuristic?

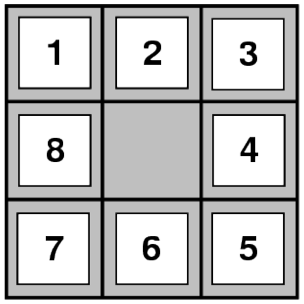
- If $h_1(n) < h_2(n) \leq h^*(n)$ for all n, h_2 is better than (**dominates**) h_1.
- **Relaxing the problem:** remove constraints to create a (much) easier problem; use the solution cost for this problem as the heuristic function
- **Combining heuristics:** take the max of several admissible heuristics: still have an admissible heuristic, and it's better!
- **Use statistical estimates to compute *h*:** may lose admissibility
- **Identify good features, then use a learning algorithm to find a heuristic function:** also may lose admissibility

In-class Exercise: Creating Heuristics

8-Puzzle

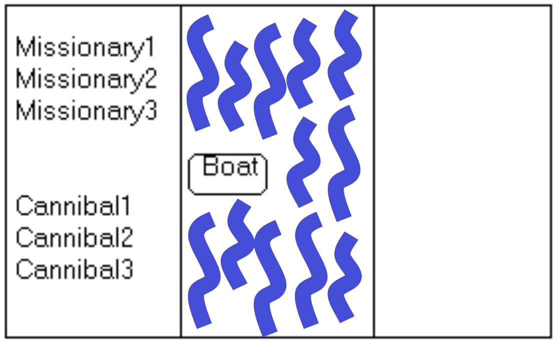


Start State

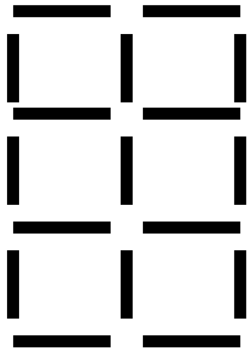


Goal State

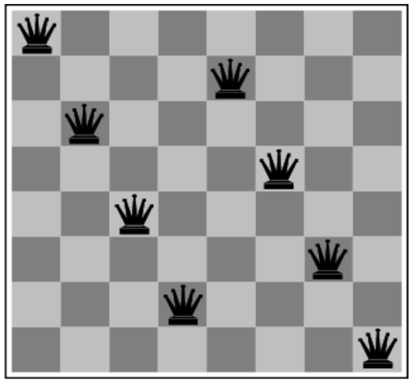
Missionaries and Cannibals



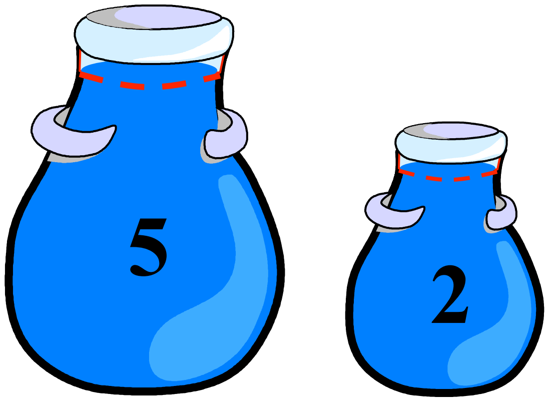
Remove 5 Sticks



N-Queens



Water Jug Problem



Route Planning

North Sea
NETHERLANDS
Zeebrugge
Oostende
Brugge
Antwerp
Gent
Mechelen
Kortrijk
Aalst
Leuven
BRUSSELS
GER.
Liège
Mons
Charleroi
Namur
Botrange
Bastogne
LUX.
FRANCE