Search

• What is search?
• Terminology: search space, strategy
• Modelling
• Uninformed search (not “intelligent”)
  • Breadth-first
  • Depth-first, some variations
• Complexity – space and time
• Heuristic search

The wolf-sheep-cabbage problem

You are on the bank of a river with a boat, a cabbage, a sheep, and a wolf. Your task is to get everything to the other side.
1. only you can handle the boat
2. only space for you and one more item
3. wolf eats sheep, sheep eats cabbage – State = YCSW

Terminology

• Graph
• Node
• Arc, edge – directed edge
• Path, cycle, connected
• Directed Graph, Acyclic graph, DAG
• Tree: root, leaf
• Family: parent, child, ancestor, descendant

Finite state acceptor / state space

S - the set of states
s0 - initial state
G - set of goal states - subset of S
F (flow) - transition function.

Variations: F is a subset of
S x S
S x Name x S - each move has a name.
A sequence of moves has a sequence of names (word).
The moves can be input (in Luger, Name is called I) or output.
S x S x Cost - each move has a cost.
Find not just any solution, but the cheapest solution.
S x Name x S x Cost

Solution

• A sequence of states s0, s1, ..., sn :
si in G, and (si,si+1) in F (0 ≤ i < n).

• A sequence of moves (word) a1, …, an:
(si,ai+1,si+1) in F (0 ≤ i < n) and si in G.

• The cost of a solution is (usually) the sum of the cost of the moves.
Modelling

- What are the states?
- How are the states represented?
- What are the moves?
- How are the moves represented?
- What search strategy? (Coming 3 lectures.)

Real problems are not toy problems

This is not a working approach:
- Construct the search space in memory
- Apply shortest path algorithm
  - search space is the part of the state space that can be reached by the search strategy.
  - The search tree is the part of the search space that is actually searched (can be unfolded as a tree).
- These are not data structures!
  - Usually only parts of the search tree are present in memory during the search.

8-, 15-, 39-puzzle

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

- $9! / 2 = 181,440$
- $16! / 2 = 1 \times 10^{13}$
- $40! / 2 = 4 \times 10^{47}$

A model with cost

Smarter modelling

A model with cost
Search
• What is search?
• Terminology: search space, strategy
• Modelling
  • Uninformed search (not “intelligent”)  
    • Breadth-first
    • Depth-first, some variations
    • Complexity — space and time
  • Heuristic search

Search tree

Search data structures
• Open nodes — your “to do” list  
  — The order is very important
• Closed nodes  
  — A set of nodes you are done with  
  — May take too much memory to keep all nodes
• Parent of each node  
  — To recreate the solution after the search
• Cost, etc.

Search algorithm pattern
Repeat
• Pick the first node N from Open
• N is a goal node? Done!
• Compute children of N  
  • Remove those that are in Open or Closed
  • Add new children of N to Open
  • Put N in Closed

The solution is the path back from N, using Parent

Breadth-first search
• Search the tree level by level
• Open is a queue

Properties
• Finds the shortest path
• Complete

Complexity
• B = branching factor
• Dg = depth of Goal (shortest path to goal)
Size of Open = B^{Dg}
Depth-first search

- Go down first
- \( \text{Open} \) is a stack

Properties

- \( B \) = branching factor
- \( D_t \) = depth of tree

Size of \( \text{Open} \) = \( B \times D_t \) ← The good news
Size of \( \text{Closed} \) = \( B^{D_t} \)

No guarantees:
- Not complete (if search space is infinite)
- Not the shortest path

Why \( \text{Closed} \)?

- To avoid loops
- To avoid double work

State space:

Is it worth it to use \( \text{Closed} \)?

Don’t use \( \text{Closed} \) in DFS!

- If there are loops, use the list of parents
- Risk of double work, but:

Repeat
- Pick the first node \( N \) from \( \text{Open} \)
- \( N \) is a goal node? Done!
- Compute children of \( N \)
  - Remove those that are in \( \text{Open} \) or \( \text{Closed} \)
  - Add new children of \( N \) to \( \text{Open} \)
  - Put \( N \) in \( \text{Closed} \)

requires that you check \( \text{Closed} \) every step!
(Hashing might help a lot, but ...)

To summarize

Depth-first search

Breadth-first
- Complete
- Finds shortest path to goal
- Exponential memory

Depth-first without \( \text{Closed} \)
- Not complete
- No guarantee
- Linear memory

Can we have the best of both?
Binding the depth

**Bounded depth-first search**
- You have an upper bound \( d \) for the shortest solution?
  - Do a depth-first search up to depth \( d \): \text{BDFS}(d)
  - Complete.
  - Linear memory.

**Iterative deepening**
- \( i = 1 \)
  - Repeat
    - \text{BDFS}(i)
    - \( i = i + 1 \)
  - Complete.
  - Finds shortest path to goal
  - Linear memory.
  - Repeated work is only \( 1/(B-1) \)

Search

- What is search?
- Terminology: search space, strategy
- Modelling
- Uninformed search (not “intelligent”)
  - Breadth-first
  - Depth-first, some variations
  - Complexity – space and time
- Heuristic search

Heuristic search

- Heuristic =
  - (Quantitative) information: good – bad
  - Approximate – can be wrong sometimes!
  - Domain/problem dependent
  - Understand the problem
  - Appear intelligent

Hill-Climbing

Each state \( S \) has a heuristic value \( h(S) \).
Higher is better!
Goal: reach the state with the highest value.

\( N = \) start state

Repeat
- Compute children \( C_1, ..., C_n \) of \( N \)
- Compute \( h(C_1), ..., h(C_n) \)
- Find maximum value \( h(C) \) for child \( C \)
- If \( h(N) > h(C) \)
  - then Stop (you’re at a top!)
  - else \( N = C \)

Uninformed - Heuristic

<table>
<thead>
<tr>
<th>Depth-first</th>
<th>Hill climbing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-first</td>
<td>Best-first search A and A*</td>
</tr>
</tbody>
</table>

Hill-Climbing

- There is no going back
- Risks to find a **local maximum**
- In large search spaces, this can be the only feasible method (**learning**).

- Some variations
  - Restart from different (random) start states
  - Simulated annealing
    (probabilistically accept bad moves)
Best-first search

- Each state $S$ has a heuristic value.
- Lower is better!
- Open is sorted best-first
- Chooses the globally best step (hill-climbing the locally best step)

The heuristic estimate

- How do we get a good estimate?
- What is a good estimate?
  – Close to the actual value ...
- Over- or underestimate?

Algorithm A

Best-first search, using the following $f$:
For a node $n$, estimate

$$f(n) \text{ [distance from start to goal via n]} = g(n) \text{ [length of shortest path to n found so far]} + h(n) \text{ [heuristic: guessed distance from n to goal]}$$
The “lamp over the bridge” example

\[ h(n) = \begin{cases} \text{if } L \text{ then “sum of YWGC”} \\ \text{else “sum of YWGC” + 4} \end{cases} \]

Compare

\[ f(n) \quad \text{[estimated distance from start to goal via n]} = \]
\[ g(n) \quad \text{[length of shortest path to n found so far]} + \]
\[ h(n) \quad \text{[heuristic: guessed distance from n to goal]} = \]
\[ f^*(n) \quad \text{[shortest distance from start to goal via n]} = \]
\[ g^*(n) \quad \text{[length of shortest path to n]} + \]
\[ h^*(n) \quad \text{[shortest distance from n to goal]} \]

Algorithm A*

• Best-first search,
• \( f(n) = g(n) + h(n) \),
• \( h(n) \leq h^*(n) \) for all nodes n. “h is optimistic”

When we close the goal, we have found the shortest path.

Proof (informal)

• Suppose the path is not optimal.
• Let n be the first node on the optimal path that is not closed.
• The path found to n is optimal: \( g(n) = g^*(n) \).
• \( f(G) = g(G) + h(G) = g(G) \geq f^*(n) = g^*(n) + h^*(n) = g(n) + h(n) = f(n) \)
• n should have been closed before G.

Monotonicity (consistency)

• Between two adjacent nodes, their heuristic value differs not more than their distance.
• and \( h(G) = 0 \).
• Monotonic → along a path, estimates go up
• Monotonic → For every node that is closed, we have the shortest path
• Monotonic → Optimistic
Example (not optimistic)

The high estimate for B blocks the shortest path.

Example (not monotonic)

C was closed, but reopened when a shorter path was found.

Example (monotonic)

Proofs

• For every node that is closed, we have the shortest path.

When we close a node n, all nodes with a lower estimate are closed (and since estimates along a path go up) all nodes on the shortest path to n are closed.

Proofs

• Along a path, estimates go up.

\[
\begin{align*}
\text{n} & \rightarrow \text{d} & \rightarrow \text{m} \\
\text{n} = g(n) + h(n) & \quad & \text{f(m)} = g(m) + h(m) \\
\text{g(m)} = g(n) + d & \quad & h(n) - h(m) \leq d \\
f(m) = g(m) + h(m) & \geq g(n)+d + h(n) - d = f(n)
\end{align*}
\]

Proofs

• Monotonic \(\rightarrow\) Optimistic

• Take the optimal path from n to G: \(n = n_1, n_2, ..., n_k = G\).

• Compare the sequences

\[
\begin{align*}
0 & = h^*(G), h^*(n_k),..., h^*(n_1), h^*(n), \\
0 & = h(G), h(n_k),..., h(n_1), h(n).
\end{align*}
\]

• The first sequence actual distances has bigger steps than the second heuristic differences

• \(h^*(n) \geq h(n)\).
The heuristic estimate

- How do we get a good estimate? domain dependent
- Over- or underestimate? underestimate
general idea: ignore some constraints
- What is a good estimate? h(n) = 0?
  - Close to the actual value ... informedness

Informedness

- For all nodes n: h₁(n) ≤ h₂(n) ≤ h*(n)
  h₂ is more informed than h₁
- Result:
  h₂ expands fewer nodes than h₁

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Example:

h₁(n) = # tiles out of place 7
h₂(n) = ∑ distance a tiles must move
0+1+4+2+1+2+3+1 = 14
h*(n) = 22

Which nodes are expanded?

- If h is monotonic:
  all nodes n with f(n) = g*(n)+h(n) < f(G)
  and some nodes where f(n)=f(G)
- If h is higher, fewer nodes are expanded.
- Trade-off:
  computing "smarter" h takes more time.

Iterative deepening A* (IDA*)

- Cutoff = h(S)
- Repeat
  - run DFS, expand node n only if f(n) ≤ Cutoff
  - Cutoff = lowest f-value of open nodes
- Until goal node is expanded