Uninformed Search

Day 1 of Search

Russel & Norvig Chap. 3
Material in part from http://www.cs.cmu.edu/~awm/tutorials

Search

- Examples of Search problems?
- The Oak Tree
- Informed versus Uninformed
  - Heuristic versus Blind

A Search Problem

- Find a path from START to GOAL
- Find the minimum number of transitions

Example

- State: Configuration of puzzle
- Transitions: Up to 4 possible moves (up, down, left, right)
- Solvable in 22 steps (average)
- But: $1.8 \times 10^5$ states ($1.3 \times 10^{12}$ states for the 15-puzzle)
  → Cannot represent set of states explicitly

Example: Robot Navigation

States = positions in the map
Transitions = allowed motions
Navigation: Going from point START to point GOAL given a (deterministic) map
Example Solution: Brushfire...

Other Real-Life Examples

Protein design
http://www.blueprint.org/proteinfolding/trades/trades_problem.html

Scheduling/Manufacturing

Route planning
http://www.ozone.ri.cmu.edu/projects/dms/dmsmain.html

Scheduling/Science
http://www.ozone.ri.cmu.edu/projects/hsts/hstsmain.html

Robot navigation
http://www.frc.ri.cmu.edu/projects/mars/dstar.html

Don't necessarily know explicitly the structure of a search problem

Other Real-Life Examples

Protein design
http://www.blueprint.org/proteinfolding/trades/trades_problem.html

Scheduling/Manufacturing

Route planning
http://www.ozone.ri.cmu.edu/projects/dms/dmsmain.html

Scheduling/Science
http://www.ozone.ri.cmu.edu/projects/hsts/hstsmain.html

Robot navigation
http://www.frc.ri.cmu.edu/projects/mars/dstar.html

Don't have a clue when you're doing well versus poorly!

10cm resolution
4km^2 ~ 4 \times 10^8 states

What we are not addressing (yet)

- Uncertainty/Chance → State and transitions are known and deterministic
- Game against adversary
- Multiple agents/Cooperation
- Continuous state space → For now, the set of states is discrete

Overview

- Definition and formulation
- Optimality, Completeness, and Complexity
- Uninformed Search
  - Breadth First Search
  - Search Trees
  - Depth First Search
  - Iterative Deepening
- Informed Search
  - Best First Greedy Search
  - Heuristic Search, A*

Don't necessarily know explicitly the structure of a search problem
A Search Problem: Square World

Formulation
- $Q$: Finite set of states
- $S \subseteq Q$: Non-empty set of start states
- $G \subseteq Q$: Non-empty set of goal states
- $\text{succs}$: function $Q \to P(Q)$
  - $\text{succs}(s) = \text{Set of states that can be reached from } s \text{ in one step}$
- $\text{cost}$: function $Q \times Q \to \text{Positive Numbers}$
  - $\text{cost}(s, s') = \text{Cost of taking a one-step transition from state } s \text{ to state } s'$
- Problem: Find a sequence $\{s_1, \ldots, s_K\}$ such that:
  1. $s_1 \in S$
  2. $s_K \in G$
  3. $s_{i+1} \in \text{succs}(s_i)$
  4. $\sum \text{cost}(s_i, s_{i+1})$ is the smallest among all possible sequences (desirable but optional)

What about actions?
- $Q$: Finite set of states
- $S \subseteq Q$: Non-empty set of start states
- $G \subseteq Q$: Non-empty set of goal states
- $\text{succs}$: function $Q \to P(Q)$
  - $\text{succs}(s) = \text{Set of states that can be reached from } s \text{ in one step}$
- $\text{cost}$: function $Q \times Q \to \text{Positive Numbers}$
  - $\text{cost}(s, s') = \text{Cost of taking a one-step transition from state } s \text{ to state } s'$
- Problem: Find a sequence $\{s_1, \ldots, s_K\}$ such that:
  - Actions define transitions from states to states.
  - Example: Square World

Example
- $Q = \{AA, AB, AC, AD, AI, BB, BC, BD, BI, \ldots\}$
- $S = \{AB\}$, $G = \{DD\}$
- $\text{succs}(AA) = \{AI, BA\}$
- $\text{cost}(s, s') = 1$ for each action (transition)

Desirable Properties
- Completeness: An algorithm is complete if it is guaranteed to find a path if one exists
- Optimality: The total cost of the path is the lowest among all possible paths from start to goal
- Time Complexity
- Space Complexity

Breadth-First Search
- Label all states that are 0 steps from $S$ to $G$
  - Call that set $V_0$
Breadth-First Search

- Label the successors of the states in $V_0$ that are not yet labelled → Set $V_1$ of states that are 1 step away from the start

- Label the successors of the states in $V_1$ that are not yet labelled → Set $V_2$ of states that are 1 step away from the start

- Stop when goal is reached in the current expansion set → goal can be reached in 4 steps

Recovering the Path

- Record the predecessor state when labeling a new state
- When I labeled GOAL, I was expanding the neighbors of $f$ so therefore $f$ is the predecessor of GOAL
- When I labeled $f$, I was expanding the neighbors of $r$ so therefore $r$ is the predecessor of $f$
- Final solution: $\{\text{START, e, r, f, GOAL}\}$

Using Backpointers

- A backpointer $\text{previous}(s)$ points to the node that stored the state that was expanded to label $s$
- The path is recovered by following the backpointers starting at the goal state
Example: Robot Navigation

States = positions in the map
Transitions = allowed motions

Navigation: Going from point START to point GOAL given a (deterministic) map

Breadth First Search

$V_0 \leftarrow S$ (the set of start states)

$\text{previous}(\text{START}) := \text{NULL}$

$k \leftarrow 0$

while (no goal state is in $V_k$ and $V_k$ is not empty) do

$V_{k+1} \leftarrow \text{empty set}$

For each state $s$ in $V_k$

For each state $s'$ in $\text{succs}(s)$

If $s'$ has not already been labeled

Set $\text{previous}(s') \leftarrow s$

Add $s'$ into $V_{k+1}$

$k \leftarrow k + 1$

if $V_k$ is empty signal FAILURE

else build the solution path thus:

Define $S_k = \text{GOAL}$ and for all $i \leq k$, define $S_i = \text{previous}(S_{i+1})$

Return path $= \{S_1, ..., S_k\}$

Properties

- BFS can handle multiple start and goal states *what does multiple start mean?*
- Can work either by searching forward from the start or backward for the goal (forward/backward chaining)
- (Which way is better?)
- Guaranteed to find the lowest-cost path in terms of number of transitions??

See maze example

Complexity

- $N = \text{Total number of states}$
- $B = \text{Average number of successors (branching factor)}$
- $L = \text{Length from start to goal with smallest number of steps}$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Breadth First Search</td>
<td>Y</td>
<td>Y, if all trans. have same cost.</td>
<td>$O(\min(N,B))$</td>
</tr>
</tbody>
</table>

Bidirectional Search

- BFS search simultaneously forward from $\text{START}$ and backward from $\text{GOAL}$
- When do the two search meet?
- What stopping criterion should be used?
- Under what condition is it optimal?
**Complexity**

- \( N \) = Total number of states
- \( B \) = Average number of successors (branching factor)
- \( L \) = Length for start to goal with smallest number of steps

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td>O(min((N,B)))</td>
<td>O(min((N,B)))</td>
</tr>
<tr>
<td>BiBFS</td>
<td></td>
<td></td>
<td>O(min((N,2B^{L/2})))</td>
<td>O(min((N,2B^{L/2})))</td>
</tr>
</tbody>
</table>

Major savings when bidirectional search is possible because \( 2B^{L/2} \ll B^{L} \)

\[ B = 10, L = 6 \rightarrow 22,200 \text{ states generated vs. ~}10^7 \]

### Counting Transition Costs Instead of Transitions

- BFS finds the shortest path in number of steps but does not take into account transition costs
- Simple modification finds the least cost path
- New field: \( g(s) = \) least cost path to \( s \) in \( k \) or fewer steps
Uniform Cost Search

- Strategy to select state to expand next
- Use the state with the smallest value of \( g(s) \) so far
- Use priority queue for efficient access to minimum \( g \) at every iteration

Priority Queue

- Priority queue = data structure in which data of the form \((\text{item}, \text{value})\) can be inserted and the item of minimum value can be retrieved efficiently
- Operations:
  - \text{Init} (PQ): Initialize empty queue
  - \text{Insert} (PQ, item, value): Insert a pair in the queue
  - \text{Pop} (PQ): Returns the pair with the minimum value
- In our case:
  - \text{item} = state \quad \text{value} = \text{current cost} \( g(s) \)

Complexity: \( O(\log(\text{number of pairs in PQ})) \) for insertion and pop operations \( \rightarrow \) very efficient

http://www.lee.kilough.com/heaps/ Knuth&Sedwick .....

---

\( PQ = \{(\text{START},0)\} \)

1. Pop the state \( s \) with the lowest path cost from \( PQ \)
2. Evaluate the path cost to all the successors of \( s \)
3. Add the successors of \( s \) to \( PQ \)

\( PQ = \{(p,1)\} \)

1. Pop the state \( s \) with the lowest path cost from \( PQ \)
2. Evaluate the path cost to all the successors of \( s \)
3. Add the successors of \( s \) to \( PQ \)

---

\( PQ = \{(d,3)\} \)

1. Pop the state \( s \) with the lowest path cost from \( PQ \)
2. Evaluate the path cost to all the successors of \( s \)
3. Add the successors of \( s \) to \( PQ \)
PQ = \{(b,4) (e,5) (c,11) (q,16)\}
1. Pop the state s with the lowest path cost from PQ
2. Evaluate the path cost to all the successors of s
3. Add the successors of s to PQ

Important: We realized that going to e through d is cheaper than going to e directly ⇒ the value of e is updated from 9 to 5 and it moves up in PQ

PQ = \{(a,6) (h,6) (c,11) (r,14) (q,16)\}
1. Pop the state s with the lowest path cost from PQ
2. Evaluate the path cost to all the successors of s
3. Add the successors of s to PQ

PQ = \{(h,6) (c,11) (r,14) (q,16)\}
1. Pop the state s with the lowest path cost from PQ
2. Evaluate the path cost to all the successors of s
3. Add the successors of s to PQ

PQ = \{(q,10) (c,11) (r,14)\}
1. Pop the state s with the lowest path cost from PQ
2. Evaluate the path cost to all the successors of s
3. Add the successors of s to PQ
1. Pop the state \( s \) with the lowest path cost from \( PQ \).
2. Evaluate the path cost to all the successors of \( s \).
3. Add the successors of \( s \) to \( PQ \).

Example: Robot Navigation

- States = positions in the map
- Transitions = allowed motions
- Navigation: Going from point START to point GOAL given a (deterministic) map

Final path: (START, d, e, h, q, r, f, GOAL)

- This path is optimal in total cost even though it has more transitions than the one found by BFS.
- What should be the stopping condition?
- Under what conditions is UCS complete/optimal?
**Complexity**

- \( N \) = Total number of states
- \( B \) = Average number of successors (branching factor)
- \( L \) = Length for start to goal with smallest number of steps
- \( Q \) = Average size of the priority queue

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td>( O(N) )</td>
<td>( O(N) )</td>
</tr>
<tr>
<td>BIBFS</td>
<td>Y</td>
<td></td>
<td>( O(N) )</td>
<td>( O(N) )</td>
</tr>
<tr>
<td>UCS</td>
<td>Y, if cost &gt; 0</td>
<td>( O(N) )</td>
<td>( O(N) )</td>
<td></td>
</tr>
</tbody>
</table>

**Limitations of BFS**

- Memory usage is \( O(B^L) \) in general
- Limitation in many problems in which the states cannot be enumerated or stored explicitly, e.g., large branching factor
- Alternative: Find a search strategy that requires little storage for use in large problems

**Philosophical Limitation**

- We cannot shoot for perfection, we want good enough…

**Depth First Search**

- General idea:
  - Expand the most recently expanded node if it has successors
  - Otherwise backup to the previous node on the current path
DFS Implementation

DFS (s)
if s = GOAL
    return SUCCESS
else
    For all s' in succs(s)
        DFS (s')
    return FAILURE

s is current state being expanded, starting with START

DFS

In a recursive implementation, the program stack keeps track of the states in the current path.

Search Tree Interpretation

• Root: START state
• Children of node containing state s: All states in succs(s)
• In the worst case the entire tree is explored \(O(B^{L_{max}})\)
• Infinite branches if there are loops in the graph!

Complexity

- \(N = \) Total number of states
- \(B = \) Average number of successors (branching factor)
- \(L = \) Length for start to goal with smallest number of steps
- \(C = \) Cost of optimal path
- \(Q = \) Average size of the priority queue
- \(L_{max} = \) Length of longest path from START to any state

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Y</td>
<td>Y, if all trans. have same cost</td>
<td>(O(\min(N,L)))</td>
<td>(O(\min(N,L)</td>
</tr>
<tr>
<td>BBFS</td>
<td>Y</td>
<td>Y, if all trans. have same cost</td>
<td>(O(\min(NL^{2+\epsilon})))</td>
<td>(O(\min(NL^{2+\epsilon})</td>
</tr>
<tr>
<td>UCS</td>
<td>Y (if cost &gt; 0)</td>
<td>Y</td>
<td>(O(\log Q \cdot \min(N,L^{2+\epsilon})))</td>
<td>(O(\min(N,L^{2+\epsilon})</td>
</tr>
<tr>
<td>DFS</td>
<td>Y (if cost &gt; 0)</td>
<td>Y</td>
<td>(O(\log Q \cdot \min(N,L^{2+\epsilon})))</td>
<td>(O(\min(N,L^{2+\epsilon})</td>
</tr>
</tbody>
</table>

For graphs without cycles
Is this a problem:
• $L_{\text{max}}$ = Length of longest path from START to any state

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Y</td>
<td></td>
<td>$O(\min(NL))$</td>
<td>$O(\min(NL))$</td>
</tr>
<tr>
<td>BIBFS</td>
<td>Y</td>
<td></td>
<td>$O(\min(NL))$</td>
<td>$O(\min(NL))$</td>
</tr>
<tr>
<td>UCS</td>
<td>Y, If cost &gt; 0</td>
<td></td>
<td>$O(\log(Q\cdot\min(NL^2))$</td>
<td>$O(\min(NL^3))$</td>
</tr>
<tr>
<td>DFS</td>
<td>$\frac{N}{2}$</td>
<td></td>
<td>$O(L^{\min})$</td>
<td>$O(BL_{\text{max}})$</td>
</tr>
</tbody>
</table>

Complexity

• $N$ = Total number of states
• $B = \text{Average number of successors (branching factor)}$
• $L = \text{Length for start to goal with smallest number of steps}$
• $C = \text{Cost of optimal path}$
• $Q = \text{Average size of the priority queue}$
• $L_{\text{max}} = \text{Length of longest path from START to any state}$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Y</td>
<td></td>
<td>$O(\min(NL))$</td>
<td>$O(\min(NL))$</td>
</tr>
<tr>
<td>BIBFS</td>
<td>Y</td>
<td></td>
<td>$O(\min(NL))$</td>
<td>$O(\min(NL))$</td>
</tr>
<tr>
<td>UCS</td>
<td>Y, If cost &gt; 0</td>
<td></td>
<td>$O(\log(Q\cdot\min(NL^2))$</td>
<td>$O(\min(NL^3))$</td>
</tr>
<tr>
<td>PCDFS</td>
<td>Y</td>
<td></td>
<td>$O(L^{\min})$</td>
<td>$O(BL_{\text{max}})$</td>
</tr>
<tr>
<td>MEMDFS</td>
<td>Y</td>
<td></td>
<td>$O(L^{\min})$</td>
<td>$O(BL_{\text{max}})$</td>
</tr>
</tbody>
</table>

DFS Limitation 1

• Need to prevent DFS from looping
• Avoid visiting the same states repeatedly

Because $L$ may be much larger than the number of states $d$ steps away from the start.

• PC-DFS (Path Checking DFS):
  – Don’t use a state that is already in the current path
• MEMDFS (Memorizing DFS):
  – Keep track of all the states expanded so far. Do not expand any state twice

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Y</td>
<td></td>
<td>$O(\min(NL))$</td>
<td>$O(\min(NL))$</td>
</tr>
<tr>
<td>BIBFS</td>
<td>Y</td>
<td></td>
<td>$O(\min(NL))$</td>
<td>$O(\min(NL))$</td>
</tr>
<tr>
<td>UCS</td>
<td>Y, If cost &gt; 0</td>
<td></td>
<td>$O(\log(Q\cdot\min(NL^2))$</td>
<td>$O(\min(NL^3))$</td>
</tr>
<tr>
<td>PCDFS</td>
<td>Y</td>
<td></td>
<td>$O(L^{\min})$</td>
<td>$O(BL_{\text{max}})$</td>
</tr>
<tr>
<td>MEMDFS</td>
<td>Y</td>
<td></td>
<td>$O(L^{\min})$</td>
<td>$O(BL_{\text{max}})$</td>
</tr>
</tbody>
</table>

DFS Limitation 2

• Need to make DFS optimal

• IDS (Iterative Deepening Search):
  – Run DFS by searching only path of length 1 (DFS stops if length of path is greater than 1)
  – If that doesn’t find a solution, try again by running DFS on paths of length 2 or less
  – If that doesn’t find a solution, try again by running DFS on paths of length 3 or less
  – …………
  – Continue until a solution is found

Depth-Limited Search
Iterative Deepening Search

- Sounds horrible: We need to run DFS many times
- Actually not a problem:
  \[ O(LB^i+(L-1)B^i+\ldots+B^i) = O(B^i) \]

- Compare \( B^i \) and \( B^{i_{\text{max}}} \)
- Optimal if transition costs are equal

**Complexity**

- \( N \): Total number of states
- \( B \): Average number of successors (branching factor)
- \( L \): Length for start to goal with smallest number of steps
- \( C \): Cost of optimal path
- \( Q \): Average size of the priority queue
- \( L_{\text{max}} \): Length of longest path from START to any state

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BiBFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCDFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEMDFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BiBFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UCS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCDFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEMDFS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Summary**

- Basic search techniques: BFS, UCS, PCDFS, MEMDFS, DFID
- Property of search algorithms: Completeness, optimality, time and space complexity
- Iterative deepening and bidirectional search ideas
- Trade-offs between the different techniques and when they might be used
Some Challenges

• Driving directions
• Robot navigation in Wean Hall
• Adversarial games
  – Tic Tac Toe
  – Chess