Search

George Konidaris
gdk@cs.brown.edu

Fall 2019

(pictures: Wikipedia)
Basic to problem solving:

• *How to take action to reach a goal?*
Choices have consequences!
Search

Formalizing the problem statement …

• Problem can be in various states.
• Start in an *initial state*.
• Have some *actions* available.
• Each *action changes state*.
• Each action has a *cost*.
• Want to reach some *goal*, minimizing cost.

Happens in simulation.

*Not* web search.
Formal Definition

Set of states $S$

Start state $s \in S$

Set of actions $A$ and action rules $a(s) \rightarrow s'$

Goal test $g(s) \rightarrow \{0, 1\}$

Cost function $C(s, a, s') \rightarrow \mathbb{R}^+$

So a search problem is specified by a tuple, $(S, s, A, g, C)$. 
Problem Statement

Find a sequence of actions $a_1, \ldots, a_n$ and corresponding states $s_1, \ldots, s_n$

such that:

\[ s_0 = s \]
\[ s_i = a_i(s_{i-1}), \quad i = 1, \ldots, n \]
\[ g(s_n) = 1 \]

while minimizing:

\[ \sum_{i=1}^{n} C(s_{i-1}, a_i, s_i) \]

minimize sum of costs - rational agent
Example

Sudoku

States: all legal Sudoku boards.

Start state: a particular, partially filled-in, board.

Actions: inserting a *valid* number into the board.

Goal test: all cells filled and no collisions.

Cost function: 1 per move.
Example

**States:** airports, times.

**Start state:** TF Green at 5pm.

**Actions:** available flights from each airport after each time.

**Goal test:** reached Tokyo by midnight tomorrow.

**Cost function:** time and/or money.
The Search Tree

Classical conceptualization of search.
The Search Tree
Important Quantities

Branching factor (*breadth*)

![Diagram showing branching factor](image)
The Search Tree

Depth
• min solution depth $m$
• depth $d$

$O(b^d)$ leaves in a tree of breadth $b$, depth $d$.

$\sum_{i=0}^{d} b^i \in O(b^d)$ total nodes in the same tree
The Search Tree

Expand the tree one node at a time.
Frontier: set of nodes in tree, but not expanded.

Key to a search algorithm: which node to expand next?
visited = {}
frontier = \{s_0\}
goal_found = false

while not goal_found:
    node = \textbf{frontier.next()}
    frontier.del(node)

    if(g(node)):
        goal_found = true
    else:
        visited.add(node)
        for child in node.children:
            if(not visited.contains(child)):
                frontier.add(child)
How to Expand?

*Uninformed strategy:*

- nothing known about likely solutions in the tree.

What to do?

- Expand deepest node (*depth-first search*)
- Expand closest node (*breadth-first search*)

Properties

- Completeness
- Optimality
- Time Complexity (*total number of nodes visited*)
- Space Complexity (*size of frontier*)
Depth-First Search

Expand deepest node
Depth-First Search

Expand deepest node
Depth-First Search

Expand deepest node
Depth-First Search

Expand deepest node
Depth-First Search

Expand deepest node
DFS: Time

\[ O(b^d - b^{d-m}) = O(b^d) \]
DFS: Space

worst case: search reaches bottom

\[ O((b - 1)d) = O(bd) \]
Depth-First Search

Properties:
• Completeness: Only for finite trees.
• Optimality: No.
• Time Complexity: $O(b^d)$
• Space Complexity: $O(bd)$

Note that when reasoning about DFS, $m$ is depth of found solution (not necessarily min solution depth).

The deepest node happens to be the one you most recently visited - easy to implement recursively OR manage frontier using LIFO queue.
Breadth-First Search

Expand shallowest node

\begin{itemize}
\item \textbf{s0}
\item \textbf{s1}
\end{itemize}
Breadth-First Search

Expand shallowest node
Breadth-First Search

Expand shallowest node
Breadth-First Search

Expand shallowest node
Breadth-First Search

Expand shallowest node
BFS: Time

\[ O(b^m) \]
BFS: Space

\[ O(b^{m+1}) \]
Breadth-First Search

Properties:
• Completeness: Yes.
• Optimality: Yes for *constant cost*.
• Time Complexity: $O(b^m)$
• Space Complexity: $O(b^{m+1})$

Manage frontier using FIFO queue.
Bidirectional Search
Bidirectional Search

Why?

\[ 2 \times O(b^{\frac{d}{2}}) \text{ is way less than } O(b^d) \]

Extra requirements:

• Must be able to invert action rules.
• Sometimes easy, sometimes hard.
• Not always unique.

When do you stop?

• Candidate solution when the frontiers intersect
• That solution may not be optimal - first must exhaust possible shortcuts.
Iterative Deepening Search

DFS: great memory cost - $O(bd)$ - but suboptimal solution.

BFS: optimal solution but horrible memory cost: $O(b^{m+1})$.

The core problems in DFS are a) not optimal, and b) not complete ... because it fails to explore other branches.

Otherwise it’s a very nice algorithm!

Iterative Deepening:
• Run DFS to a fixed depth $z$.
• Start at $z=1$. If no solution, increment $z$ and rerun.
IDS

run DFS to this depth
IDS

How can that be a good idea? *It duplicates work.*

Optimal for constant cost! *Proof?*

Also!
- Low memory requirement (equal to DFS).
- Not many more nodes expanded than BFS. (About twice as many for binary tree.)
IDS

visited \(m + 1\) times

visited \(m\) times

\[
\ldots
\]
\[ \sum_{i=0}^{m} b^i(m - i + 1) = \frac{b(b^{m+1} - m - 2) + m + 1}{(b - 1)^2} \]

\# revisits

\# nodes at level \( i \)

BFS worst case:

\[ \frac{b^{m+1} - 1}{b - 1} \]
IDS

Key Insight:
• Many more nodes at depth $m+1$ than at depth $m$.

MAGIC.

“In general, iterative deepening search is the preferred uninformed search method when the state space is large and the depth of the solution is unknown.” (R&N)
Uninformed Searches So Far

Simple strategy for choosing next node:

- Choose the shallowest one (breadth-first)
- Choose the deepest one (depth-first)

Neither guaranteed to find the least-cost path, in the case where action costs are not uniform.

What if we chose the one with lowest cost?
Uniform-Cost

Order the nodes in the frontier by *cost-so-far*
  • Cost from the start state to that node.

Open the next node with the smallest cost-so-far
  • Optimal solution
  • Complete (provided no negative costs)
Uniform-Cost

Expand cheapest node
Use *whole path* cost
Uniform-Cost

Expand cheapest node
Use whole path cost
Uniform-Cost

Expand cheapest node
Use whole path cost
Uniform-Cost

Expand cheapest node
Use *whole path* cost

```
s0 --5-- s1
  |        |        |
  v        v        v
s3 --4-- s4 --7-- s7 --3-- s2
  |        |        |        |
  v        v        v        v
s5 --6-- s6 --5-- s4 --7-- s7
      |        |        |        |
      v        v        v        v
s5 ---- s6 ---- s7 ---- s8
```

- s0
- s1
- s2
- s3
- s4
- s5
- s6
- s7
- s8

Costs:
- s0 -> s1: 5
- s1 -> s4: 7
- s2 -> s7: 3
- s2 -> s8: 9
- s3 -> s4: 4
- s3 -> s5: 6
- s3 -> s6: 5
Informed Search

What if we *know something* about the search?

How should we include that knowledge? In what form should it be expressed to be useful?
What Does Uniform Cost Suggest?

The *cost-so-far* tells us how much it cost to get to a node.

- Go to cheapest nodes first.

What remains?

_{Total cost} = \text{cost-so-far} + \text{cost-to-go}_

**Cost-so-far**: cost from start to node.

**Cost-to-go**: cost from node to goal.
Informed Search

Key idea: heuristic function.

- $h(s)$ - estimates cost-to-go
  - Cost to go from state to solution.
  - Estimates $h^*(s)$ - true cost-to-go.
  - $h(s) = 0$ if $s$ is a goal.

- Problem specific (hence informed)
Greed

What if we expand the node with lowest $h(s)$?
Informed Search: A*

A* algorithm:

• $g(s)$ - cost so far (start to $s$).
• Expand $s$ that minimizes $g(s) + h(s)$ both
• Manage frontier as priority queue.

• Admissible heuristic: never overestimates cost.
  $$h(s) \leq h^*(s)$$

• $h(s) = 0$ if $s$ is a goal state, so $g(s) + h(s) = c(s)$

• If $h$ is admissible, A* finds optimal solution.
• If $h(s)$ is exact, runs in $O(bd)$ time.
Admissible Heuristics

Optimality:
Proof by contradiction
Proof

Assume:

\[ g(s_a) > g(s_{opt}) \]

But if \( s_a \) was opened before \( s_b \) then:

\[ g(s_a) + h(s_a) \leq g(s_b) + h(s_b) \]

But if \( h \) is admissible then:

\[ g(s_b) + h(s_b) \leq g(s_b) + h^*(s_b) = g(s_{opt}) \]

\[ \vdots \]

…But then:

\[ g(s_a) \leq g(s_b) + h(s_b) \leq g(s_{opt}) \]
Example Heuristic
More on Heuristics

Ideal heuristics:
• Fast to compute.
• Close to real costs.

Some programs *automatically generate* heuristics.