Robot Programming with Lisp

6. Search Algorithms

Gayane Kazhoyan
(Stuart Russell, Peter Norvig)

Institute for Artificial Intelligence
University of Bremen

November 22nd, 2018
Contents
Problem Definition

Uninformed search strategies
- BFS
- Uniform-Cost
- DFS
- Depth-Limited
- Iterative Deepening

Informed Search
- Greedy
- A*
- Heuristics
- Hill-climbing aka gradient ascent/descent
- Simulated annealing

Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)
November 22nd, 2018
Problem types

Deterministic, fully observable $\implies$ *single-state problem*
Agent knows exactly which state it will be in.
Solution is a sequence of actions

Deterministic, non-observable $\implies$ *conformant problem*
Agent may have no idea where it is.
Solution (if any) is a sequence of actions

Nondeterministic, partially observable $\implies$ *contingency problem*
must perceive the world during execution
solution is a contingent plan or a policy
often replan during execution

Unknown state space $\implies$ *exploration problem* ("online")
Example: vacuum world

Single-state, start in #5.
Solution? [Right, Vacuum]

Conformant, start in \{1, 2, 3, 4, 5, 6, 7, 8\}
e.g., Right goes to \{2, 4, 6, 8\}.
Solution? [Right, Vacuum, Left, Vacuum]

Contingency, start in #5
Vacuum can dirty a clean carpet.
Local sensing only at current location.
Solution? [Right, if dirt then Vacuum]

Problem Definition Uninformed search strategies Informed Search Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)
November 22nd, 2018
Single-state problem formulation

A problem is defined by four items:

- **initial state**
- **operators** (or successor function $S(x)$)
  e.g., Vacuum($x$) $\rightarrow$ clean room
- **goal test**
- **path cost** (additive)
  e.g., sum of distances, number of operators executed, etc.

A solution is a sequence of operators leading from the initial state to a goal state
Example: The 8-puzzle

Start State

5
4

6
1
8

7
3
2

Goal State

1
2
3

8
4

7
6
5

states ?
operators ?
goal test ?
path cost ?

Problem Definition
Uninformed search strategies
Informed Search
Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)
November 22nd, 2018

Robot Programming with Lisp
Example: The 8-puzzle

states: integer locations of tiles (ignore intermediate positions)
operators: move blank left, right, up, down (ignore unjamming etc.)
goal test: current state = goal state
path cost: 1 per move
Example: vacuum world state space graph

states ?
operators ?
goal test ?
path cost ?

Problem Definition | Uninformed search strategies | Informed Search | Organizational
--- | --- | --- | ---
Gayane Kazhoyan (Stuart Russell, Peter Norvig)
November 22\textsuperscript{nd}, 2018

Robot Programming with Lisp

8
Example: vacuum world state space graph

states: integer dirt and robot locations (ignore dirt amounts)
operators: Left, Right, Vacuum
goal test: no dirt in current state
path cost: 1 per operator
Example: robotic assembly

states ?
operators ?
goal test ?
path cost ?
Example: robotic assembly

**states**: real-valued coordinates of robot joint angles and parts of the object to be assembled

**operators**: continuous motions of robot joints

**goal test**: assembly object is complete

**path cost**: time to execute
Search algorithms

Basic idea:
offline, simulated exploration of state space
by generating successors of already-explored states
(a.k.a. expanding states)

function General-Search( problem, strategy) returns a solution, or failure
initialize the search tree using the initial state of problem
loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return corresponding solution
    else expand the node and add the resulting nodes to the search tree
end
General search example

Problem Definition
Uninformed search strategies
Informed Search
Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)
November 22nd, 2018
General search example

Arad
General search example

Problem Definition  Uninformed search strategies  Informed Search  Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)  November 22\textsuperscript{nd}, 2018  

Robot Programming with Lisp
General search example
General search example
Implementation of search algorithms

```
function General-Search( problem, Queuing-Fn) returns a solution, or failure

    nodes ← Make-Queue(Make-Node(Initial-State[problem]))
    loop do
        if nodes is empty then return failure
        node ← Remove-Front(nodes)
        if Goal-Test[problem] applied to State(node) succeeds then return node
        nodes ← Queuing-Fn(nodes, Expand(node, Operators[problem]))
    end
```

Problem Definition
Uninformed search strategies
Informed Search
Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)
November 22nd, 2018

Robot Programming with Lisp
Implementation contd: states vs. nodes

A *state* is a (representation of) a physical configuration
A *node* is a data structure constituting part of a search tree
includes *parent*, *children*, *depth*, *path cost* \( g(x) \)
*States* do not have parents, children, depth, or path cost!

The Expand function creates new nodes, filling in the various fields and using the Operators (or SuccessorFn) of the problem to create the corresponding states.
Search strategies

A strategy is defined by picking the order of node expansion.

Strategies are evaluated along the following dimensions:

- **completeness**—does it always find a solution if one exists?
- **time complexity**—number of nodes generated/expanded
- **space complexity**—maximum number of nodes in memory
- **optimality**—does it always find a least-cost solution?

Time and space complexity are measured in terms of:

- $b$ — maximum branching factor of the search tree
- $d$ — depth of the least-cost solution
- $m$ — maximum depth of the state space (may be $\infty$)
Contents

Problem Definition

Uninformed search strategies
  BFS
  Uniform-Cost
  DFS
  Depth-Limited
  Iterative Deepening

Informed Search
  Greedy
  A*
  Heuristics
  Hill-climbing aka gradient ascent/descent
  Simulated annealing

Organizational
Uninformed search strategies

*Uninformed* strategies use only the information available in the problem definition.

Uninformed search strategies are:

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search
Breadth-first search
Expand shallowest unexpanded node

Implementation:
QueueingFn = put successors at end of queue (FIFO queue)
Breadth-first search
Breadth-first search
Breadth-first search
Properties of breadth-first search

Complete?
Time?
Space?
Optimal?
Properties of breadth-first search

**Complete:** Yes
**Time:** 1 + b + b^2 + b^3 + \ldots + b^d = O(b^d), i.e., exponential in d
**Space:** O(b^d) (keeps every node in memory)
**Optimal:** Yes (if cost = 1 per step); not optimal in general

*Space* is the big problem; can easily generate nodes at *N* MB/sec.
Problem Definition

Uninformed search strategies

Informed Search

Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)
November 22nd, 2018

Robot Programming with Lisp
Uniform-cost search
Expand least-cost unexpanded node

Implementation:
QueueingFn = insert in order of increasing path cost (FIFO queue)
Uniform-cost search

Problem Definition

Uninformed search strategies

Informed Search

Organizational
Uniform-cost search
Uniform-cost search
Properties of uniform-cost search

Complete?  
Time?  
Space?  
Optimal?
Properties of uniform-cost search

Complete: Yes, if step cost $\geq \epsilon$
Time: # of nodes with $g \leq$ cost of optimal solution
Space: # of nodes with $g \leq$ cost of optimal solution
Optimal: Yes

$g(n)$ is the cost of the path up to node $n$. 
Depth-first search

Expand deepest unexpanded node

Implementation:
QueueingFn = insert successors at front of queue (LIFO)
Depth-first search

Problem Definition

Uninformed search strategies

Informed Search

Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)
November 22nd, 2018

Robot Programming with Lisp
Depth-first search

- 
- 
- 

Problem Definition

Uninformed search strategies

Informed Search

Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)

November 22nd, 2018

Robot Programming with Lisp

38
Depth-first search

I.e., depth-first search can perform infinite cyclic excursions. Need a finite, non-cyclic search space (or repeated-state checking).
DFS on a depth-3 binary tree
DFS on a depth-3 binary tree
DFS on a depth-3 binary tree
DFS on a depth-3 binary tree
DFS on a depth-3 binary tree
DFS on a depth-3 binary tree, contd.
DFS on a depth-3 binary tree
DFS on a depth-3 binary tree
DFS on a depth-3 binary tree
Properties of depth-first search

Complete?  
Time?  
Space?  
Optimal?
Properties of depth-first search

**Complete**: No: fails in infinite-depth spaces, spaces with loops
   ⇒ modify to avoid repeated states along path.
Complete in finite spaces

**Time**: $O(b^m)$: terrible if $m$ is much larger than $d$
but if solutions are dense, may be much faster than breadth-first

**Space**: $O(bm)$, i.e., linear space!

**Optimal**: No
Depth-limited search

Depth-limited search = depth-first search with depth limit /

Implementation:
Nodes at depth / have no successors
Iterative deepening search

```plaintext
function Iterative-Deepening-Search( problem) returns a solution sequence
inputs: problem, a problem

   for depth ← 0 to ∞ do
      result ← Depth-Limited-Search( problem, depth)
      if result ≠ cutoff then return result
   end
```

Problem Definition

Uninformed search strategies

Informed Search

Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)

Robot Programming with Lisp

November 22\textsuperscript{nd}, 2018
<table>
<thead>
<tr>
<th>Problem Definition</th>
<th>Uninformed search strategies</th>
<th>Informed Search</th>
<th>Organizational</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gayane Kazhoyan (Stuart Russell, Peter Norvig)</td>
<td>Iterative deepening search ( l = 0 )</td>
<td></td>
<td>Robot Programming with Lisp</td>
</tr>
</tbody>
</table>

November 22\textsuperscript{nd}, 2018
Iterative deepening search \( l = 1 \)
Iterative deepening search \( l = 1 \)
Iterative deepening search $l = 2$
Iterative deepening search $I = 2$
Iterative deepening search \( l = 2 \)
Iterative deepening search $l = 2$
Iterative deepening search \( l = 2 \)
Properties of iterative deepening search

Complete?
Time?
Space?
Optimal?
Properties of iterative deepening search

Complete: Yes
Time: \((d + 1)b^0 + db^1 + (d - 1)b^2 + \ldots + b^d = O(b^d)\)
Space: \(O(bd)\)
Optimal: Yes, if step cost = 1

Can be modified to explore uniform-cost tree.

Iterative deepening search uses only linear space and not much more time than other uninformed algorithms.
Contents

Problem Definition

Uninformed search strategies
  BFS
  Uniform-Cost
  DFS
  Depth-Limited
  Iterative Deepening

Informed Search
  Greedy
  A*
  Heuristics
  Hill-climbing aka gradient ascent/descent
  Simulated annealing

Organizational
Informed search

Idea: use an evaluation function for each node as an estimate of “desirability”

⇒ Expand most desirable unexpanded node

Implementation:
QueueingFn = insert successors in decreasing order of desirability

Informed search algorithms are:

• greedy search
• A* search
Problem Definition

Uninformed search strategies

Informed Search

Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)

November 22nd, 2018
Greedy search

Evaluation function $h(n)$ (heuristic)

$= \text{estimate of cost from } n \text{ to goal}$

E.g., $h_{SLD}(n) = \text{straight-line distance from } n \text{ to Bucharest}$

Greedy search expands the node that \textit{appears} to be closest to goal
Greedy search example
Greedy search example [2]
Greedy search example [3]

Problem Definition
Uninformed search strategies
Informed Search
Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)
November 22nd, 2018

Robot Programming with Lisp
69
Greedy search example [4]
Properties of greedy search

Complete?
Time?
Space?
Optimal?
Properties of greedy search

**Complete:** No – can get stuck in loops, e.g.,

\[ \text{iasi} \rightarrow \text{Neamt} \rightarrow \text{iasi} \rightarrow \text{Neamt} \rightarrow \ldots \]

Complete in finite space with repeated-state checking.

**Time:** \( O(b^m) \), but a good heuristic can give dramatic improvement

**Space:** \( O(b^m) \) — keeps all nodes in memory

**Optimal:** No
A* search

Idea: avoid expanding paths that are already expensive

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

$g(n) =$ cost so far to reach $n$
$h(n)$ = estimated cost to goal from $n$
$f(n)$ = estimated total cost of path through $n$ to goal

A* search uses an admissible heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from $n$.

E.g., $h_{SLD}(n)$ never overestimates the actual road distance

**Theorem:** A* search is optimal
A* search example

Arad

Problem Definition
Uninformed search strategies
Informed Search
Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)
November 22nd, 2018
A* search example [2]
A* search example [3]
A* search example [4]

Problem Definition

Uninformed search strategies

Informed Search

Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)
November 22nd, 2018

Robot Programming with Lisp

77
A* search example [5]
A* search example [6]
Optimality of A* (standard proof)

Suppose some suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.

$$f(G_2) = g(G_2) \quad \text{since} \quad h(G_2) = 0$$
$$> g(G_1) \quad \text{since} \quad G_2 \text{ is suboptimal}$$
$$\geq f(n) \quad \text{since} \quad h \text{ is admissible}$$

Since $f(G_2) > f(n)$, A* will never select $G_2$ for expansion
Optimality of A* (more useful)

Lemma: A* expands nodes in order of increasing $f$ value
Gradually adds “$f$-contours” of nodes (cf. breadth-first adds layers)
Contour $i$ has all nodes with $f = f_i$, where $f_i < f_{i+1}$
Properties of $A^*$

Complete?  
Time?  
Space?  
Optimal?
Properties of A*

**Complete:** Yes, unless there are infinitely many nodes with \( f \leq f(G) \)

**Time:** Exponential in \([\text{relative error in } h \times \text{length of soln.}]\)

**Space:** Keeps all nodes in memory

**Optimal:** Yes — cannot expand \( f_{i+1} \) until \( f_i \) is finished
Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n) =$ number of misplaced tiles
- $h_2(n) =$ total Manhattan distance

(i.e., no. of squares from desired location of each tile)

Start State

\[
\begin{array}{ccc}
5 & 4 & \text{Gray}\\
6 & 1 & 8 \\
7 & 3 & 2 \\
\end{array}
\]

Goal State

\[
\begin{array}{ccc}
1 & 2 & 3 \\
8 & \text{Gray} & 4 \\
7 & 6 & 5 \\
\end{array}
\]

$h_1(S) =$ ?

$h_2(S) =$ ?

Problem Definition | Uninformed search strategies | Informed Search | Organizational

Gayane Kazhoyan (Stuart Russell, Peter Norvig)

November 22nd, 2018
Admissible heuristics

E.g., for the 8-puzzle:

\[ h_1(n) = \text{number of misplaced tiles} \]

\[ h_2(n) = \text{total Manhattan distance} \]

(i.e., no. of squares from desired location of each tile)

\[ h_1(S) = 7 \]

\[ h_2(S) = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18 \]
Dominance

If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible) then $h_2$ dominates $h_1$ and is better for search

Typical search costs:

$d = 14$  $\text{IDS} = 3,473,941$ nodes
$A^*(h_1) = 539$ nodes
$A^*(h_2) = 113$ nodes

$d = 14$  $\text{IDS} = \text{too many nodes}$
$A^*(h_1) = 39,135$ nodes
$A^*(h_2) = 1,641$ nodes
Relaxed problems

Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem.

If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.

If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.

Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.
Example: Travelling Salesperson Problem

Find the shortest tour that visits each city exactly once

Minimum spanning tree heuristic can be computed in $O(n^2)$ and is a lower bound on the shortest (open) tour.
Example: \( n \)-queens

Put \( n \) queens on an \( n \times n \) board with no two queens on the same row, column, or diagonal
Hill-climbing (or gradient ascent/descent)

“Like climbing Everest in thick fog with amnesia”

```
function Hill-Climbing( problem) returns a solution state
  inputs: problem, a problem
  local variables: current, a node
                  next, a node
  current ← Make-Node(Initial-State[problem])
  loop do
    next ← a highest-valued successor of current
    if Value[next] < Value[current] then return current
    current ← next
  end
```
Hill-climbing contd.

Problem: depending on initial state, can get stuck on local maxima.
Artificial Intelligence

Simulated annealing

Idea: escape local maxima by allowing some “bad” moves
*but gradually decrease their size and frequency*

```latex
\begin{function}
\textbf{Simulated-Annealing}( \textit{problem}, \textit{schedule} ) \textbf{returns} a solution state
\begin{itemize}
\item \textbf{inputs}: \textit{problem}, a problem
\item \textit{schedule}, a mapping from time to “temperature”
\item \textbf{local variables}: \textit{current}, a node
\item \textit{next}, a node
\item \textit{T}, a “temperature” controlling the probability of downward steps
\end{itemize}
\textbf{steps}
\begin{itemize}
\item \textit{current} $\leftarrow$ \textbf{Make-Node}(Initial-State[\textit{problem}])
\item \textbf{for} \textit{t} $\leftarrow$ 1 to $\infty$ \textbf{do}
\item \hspace{1em} \textit{T} $\leftarrow$ \textit{schedule}[\textit{t}]
\item \hspace{1em} \textbf{if} \textit{T}=0 \textbf{then} \textbf{return} \textit{current}
\item \hspace{1em} \textit{next} $\leftarrow$ a randomly selected successor of \textit{current}
\item \hspace{1em} \textit{\Delta E} $\leftarrow$ Value[\textit{next}] $-$ Value[\textit{current}]
\item \hspace{1em} \textbf{if} \textit{\Delta E} $>$ 0 \textbf{then} \textit{current} $\leftarrow$ \textit{next}
\item \hspace{1em} \textbf{else} \textit{current} $\leftarrow$ \textit{next} only with probability $e^{\Delta E/\textit{T}}$
\end{itemize}
\end{function}
```
Properties of simulated annealing

At fixed “temperature” $T$, state occupation probability reaches Boltzmann distribution:

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

$T$ decreased slowly enough $\Rightarrow$ always reach best state.

Is this necessarily an interesting guarantee?

Devised by Metropolis et al., 1953, for physical process modelling
Widely used in VLSI layout, airline scheduling, etc.
Contents

Problem Definition

Uninformed search strategies
   BFS
   Uniform-Cost
   DFS
   Depth-Limited
   Iterative Deepening

Informed Search
   Greedy
   A*
   Heuristics
   Hill-climbing aka gradient ascent/descent
   Simulated annealing

Organizational
Links

• MIT online course on AI (available for free):

• Original version of these slides used at Berkeley by Russel in his AI course, based on the AI book of Norvig and Russel:
  http://aima.eecs.berkeley.edu/slides-pdf/
Info Summary

- Assignment code: REPO/assignment_6/src/*.lisp
- Assignment points: 7 points
- Assignment due: 28.11, Wednesday, 23:59 AM German time
- Next class: 29.11, 14:15
- Next class topic: introduction to ROS.
  (Make sure your ROS and roslisp_repl are working.)
Thanks for your attention!