Lecture 4 of 42

State Spaces, Graphs, Uninformed (Blind) Search: ID-DFS, Bidirectional, UCS/B&B Discussion: Term Projects 4 of 5

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course page: http://snipurl.com/v9v3
Course web site: http://www.kddresearch.org/Courses/CIS730
Instructor home page: http://www.cis.ksu.edu/~bhsu

Reading for Next Class:

Sections 3.5 – 3.7, p. 81 – 88; 4.1 – 4.2, p. 94 - 109, Russell & Norvig 2nd ed.
Instructions for writing project plans, submitting homework

Lecture Outline

- Reading for Next Class: Sections 3.5 – 3.7, 4.1 – 4.2, R&N 2e
- Past Week: Intelligent Agents (Ch. 2), Blind Search (Ch. 3)
  - Basic search frameworks: discrete and continuous
  - Tree search intro: nodes, edges, paths, depth
  - Depth-first search (DFS) vs. breadth-first search (BFS)
  - Completeness and depth-limited search (DLS)
- Coping with Time and Space Limitations of Uninformed Search
  - Depth-limited and resource-bounded search (anytime, anyspace)
  - Iterative deepening (ID-DFS) and bidirectional search
- Project Topic 4 of 5: Natural Lang. Proc. (NLP) & Info. Extraction
- Preview: Intro to Heuristic Search (Section 4.1)
  - What is a heuristic?
  - Relationship to optimization, static evaluation, bias in learning
  - Desired properties and applications of heuristics

Problem-Solving Agents

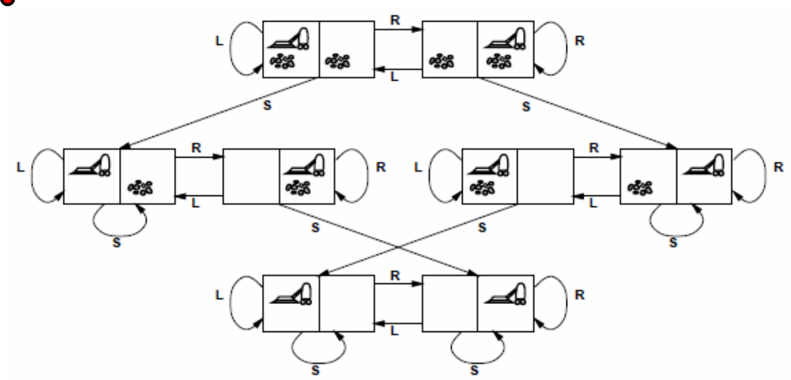
Restricted form of general agent:

```
function SIMPLE-PROBLEM-SOLVING-AGENT( percept) returns an action
   static: seq, an action sequence, initially empty
           state, some description of the current world state
           goal, a goal, initially null
           problem, a problem formulation

   state ← UPDATE-STATE(state, percept)
   if seq is empty then
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH( problem)
   action ← RECOMMENDATION(seq, state)
   seq ← REMAINDER(seq, state)
   return action
```

Note: this is offline problem solving; solution executed "eyes closed."
Online problem solving involves acting without complete knowledge.

State Space Graph: Vacuum World



states??: integer dirt and robot locations (ignore dirt amounts etc.)

actions??: *Left*, *Right*, *Suck*, *NoOp*

goal test??: no dirt

path cost??: 1 per action (0 for *NoOp*)

State Space Example: Vacuum World

Single-state, start in #5. Solution??

[$Right, Suck$]

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

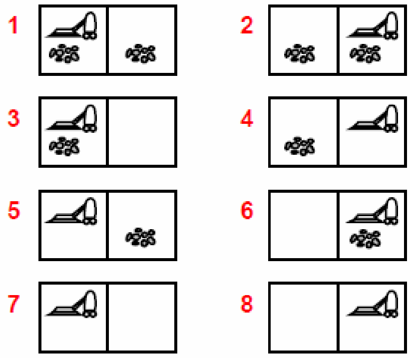
[$Right, Suck, Left, Suck$]

Sensorless

Contingency, start in #5
Murphy's Law: $Suck$ can dirty a clean carpet
Local sensing: dirt, location only.
Solution??

[$Right$, **if** $dirt$ **then** $Suck$]

Conditional



Based on slide © 2003 S. Russell & P. Norvig. Reused with permission.

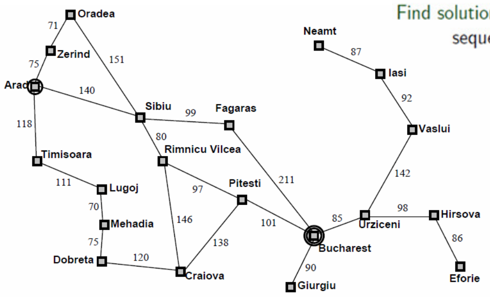
Graph Search Example: Route Planning

On holiday in Romania; currently in Arad.
Flight leaves tomorrow from Bucharest

Formulate goal:
be in Bucharest

Formulate problem:
states: various cities
actions: drive between cities

Find solution:
sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest



Edge weights
based on actual driving distance

Based on slides © 2003 S. Russell & P. Norvig. Reused with permission.

Single-State Problem Formulation

A problem is defined by four items:

initial state e.g., "at Arad"

successor function $S(x)$ = set of action–state pairs
e.g., $S(Arad) = \{\langle Arad \rightarrow Zerind, Zerind \rangle, \ldots\}$

goal test, can be
explicit, e.g., x = "at Bucharest"
implicit, e.g., $NoDirt(x)$

path cost (additive)
e.g., sum of distances, number of actions executed, etc.
$c(x, a, y)$ is the step cost, assumed to be ≥ 0

A solution is a sequence of actions
leading from the initial state to a goal state

Selecting a State Space

Real world is absurdly complex
⇒ state space must be **abstracted** for problem solving

(Abstract) state = set of real states

(Abstract) action = complex combination of real actions
e.g., "Arad $\longrightarrow$ Zerind" represents a complex set
of possible routes, detours, rest stops, etc.
For guaranteed realizability, **any** real state "in Arad"
must get to some real state "in Zerind"

(Abstract) solution =
set of real paths that are solutions in the real world

Each abstract action should be "easier" than the original problem!

General Search Algorithm: Review

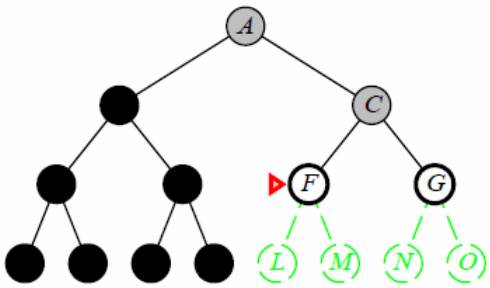
- **function** ***General-Search*** **(*problem, strategy*)**
  returns a solution or failure
  - **initialize search tree using initial state of *problem***
  - **loop do**
    - **if there are no candidates for expansion then return failure**
    - **choose leaf node for expansion according to *strategy***
    - **If node contains a goal state then return corresponding solution**
    - **else expand node and add resulting nodes to search tree**
  - **end**
- **Note: Downward Function Argument (Funarg) *strategy***
- **Implementation of *General-Search***
  - **Rest of Chapter 3, Chapter 4, R&N**
  - **See also:**
    - **Ginsberg (handout in CIS library today)**
    - **Rich and Knight**
    - **Nilsson: *Principles of Artificial Intelligence***

Depth-First Search: Review

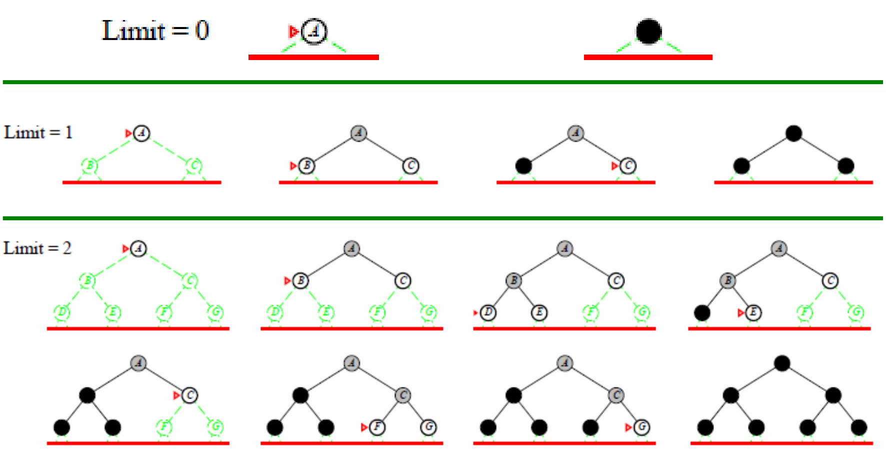
Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front



Iterative Deepening Search (ID-DFS): Example



Iterative Deepening Search (ID-DFS): Properties

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

Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$
$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100$$

IDS does better because other nodes at depth d are not expanded

BFS can be modified to apply goal test when a node is generated

Comparison of Search Strategies for Algorithms Thus Far

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

Bidirectional Search: Review

- **Intuitive Idea**
  - Search "from both ends"
  - Caveat: what does it mean to "search backwards from solution"?
- **Analysis**
  - Solution depth (in levels from root, i.e., edge depth): d
  - Analysis
    - b^i nodes generated at level i
    - At least this many nodes to test
    - Total: $\Sigma_I b^i = 1 + b + b^2 + \ldots + b^{d/2} = O(b^{d/2})$
- **Worst-Case Space Complexity:** $O(b^{d/2})$
- **Properties**
  - Convergence: suppose b, l finite and $l \geq d$
    - Complete: guaranteed to find *a* solution
    - Optimal: guaranteed to find minimum-depth solution
  - Worst-case time complexity is square root of that of BFS

Uniform-Cost Search (aka Branch & Bound) and Heuristics

- **Previously: Uninformed (Blind) Search**
  - No heuristics: only $g(n)$ used
  - Breadth-first search (BFS) and variants: uniform-cost, bidirectional
  - Depth-first search (DFS) and variants: depth-limited, iterative deepening
- **Heuristic Search**
  - Based on $h(n)$ – *estimated cost of path to goal* ("remaining path cost")
    - ⇨ h – heuristic function
    - ⇨ g: node $\rightarrow \mathbb{R}$; h: node $\rightarrow \mathbb{R}$; f: node $\rightarrow \mathbb{R}$
  - Using h
    - ⇨ h only: greedy (*aka* myopic) informed search
    - ⇨ $f = g + h$: (some) hill-climbing, A/A*
- **Uniform-Cost (Branch and Bound) Search**
  - Originates from operations research (OR)
  - Special case of heuristic search: treat as $h(n) = 0$
  - Sort candidates by $g(n)$

Heuristic Search [1]: Terminology

- **Heuristic Function**
  - Definition: $h(n)$ = estimated cost of *cheapest* path from state at node n to *a* goal state
  - Requirements for h
    - ⇨ In general, any magnitude (ordered measure, admits comparison)
    - ⇨ $h(n) = 0$ *iff* n is goal
  - For A/A*, iterative improvement: want
    - ⇨ h to have same type as g
    - ⇨ Return type to *admit addition*
  - Problem-specific (domain-specific)
- **Typical Heuristics**
  - Graph search in Euclidean space

    $h_{SLD}(n)$ = straight-line distance to goal
  - Discussion (important): *Why is this good?*

Heuristic Search [2]: Background

- **Origins of Term**
  - *Heuriskein* – to find (to discover)
  - *Heureka* ("I have found it") – attributed to Archimedes
- **Usage of Term**
  - Mathematical logic in problem solving
    - Polyà [1957]
    - Methods for discovering, inventing problem-solving techniques
    - Mathematical proof derivation techniques
  - Psychology: "rules of thumb" used by humans in problem-solving
  - Pervasive through history of AI
    - e.g., Stanford Heuristic Programming Project
    - One origin of rule-based (expert) systems
- **General Concept of Heuristic (A Modern View)**
  - Standard (rule, quantitative measure) used to *reduce search*
  - "As opposed to exhaustive blind search"
  - Compare (later): *inductive bias* in machine learning

Best-First Search [1]: Evaluation Function

- **Recall: *General-Search***
- **Applying Knowledge**
  - In problem representation (state space specification)
  - At *Insert*(), *aka Queueing-Fn*()
  - Determines node to expand next
- **Knowledge representation (KR)**
  - Expressing knowledge symbolically/numerically
  - Objective
  - Initial state
  - State space (operators, successor function)
  - Goal test: $h(n)$ – *part of* (heuristic) evaluation function

Best-First Search [2]: Characterization of Algorithm Family

- **Best-First: Family of Algorithms**
  - Justification: using only *g* doesn't *direct search toward goal*
  - Nodes ordered
  - Node with best evaluation function (e.g., *h*) expanded first
  - Best-first: any algorithm with this property (*NB*: not just using *h* alone)
- **Note on "Best"**
  - Refers to "apparent best node"
    - ⇨ based on eval function
    - ⇨ applied to current frontier
  - Discussion: when is best-first not really best?

Best-First Search [3]: Implementation

- **function *Best-First-Search* (*problem*, *Eval-Fn*) returns solution sequence**
  - inputs: *problem,* specification of problem (structure or class)
    Eval-Fn, an evaluation function
  - *Queueing-Fn* ← function that orders nodes by *Eval-Fn*
    - ⇨ Compare: *Sort* with comparator function <
    - ⇨ Functional abstraction
  - return *General-Search* (*problem*, *Queueing-Fn*)
- **Implementation**
  - Recall: priority queue specification
    - ⇨ *Eval-Fn*: node $\rightarrow \mathbb{R}$
    - ⇨ *Queueing-Fn* ≡ *Sort-By*: node list → node list
  - Rest of design follows *General-Search*
- **Issues**
  - General family of greedy (*aka* myopic, i.e., nearsighted) algorithms
  - Discussion: What guarantees do we want on $h(n)$? What preferences?

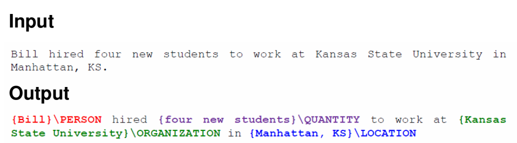
Next Topic: More on Informed Search

- Branch-and-Bound Search
- Heuristics for *General-Search* Function of *Problem-Solving-Agent*
  - Informed (heuristic) search: heuristic definition, development process
  - Best-First Search
    - Greedy
    - A/A*
    - Admissibility property
  - Developing good heuristics
    - Humans
    - Intelligent systems (automatic derivation): case studies and principles
- Constraint Satisfaction Heuristics
- This Week: More Search Basics
  - Memory bounded, iterative improvement (gradient, Monte Carlo search)
  - Introduction to game tree search

Project Topic 4 of 5: NLP and Information Extraction (IE)

Machine Translation

© 2009 Google, Inc.



Named Entity Recognition

Conversational Agent

© 1997 Institut Jozef Stefan (IJS), Slovenia

Terminology

- State Space Search
- Search Types: Uninformed ("Blind") vs. Informed ("Heuristic")
- Basic Search Algorithms
  - British Museum (depth-first *aka* DFS)
  - Breadth-First *aka* BFS
  - Depth-Limited Search (DLS)
- Refinements
  - Iterative-deepening DFS (ID-DFS)
  - Bidirectional (as adaptation of BFS or ID-DFS)
- Cost, $c(n_1, n_2)$ and Cumulative Path Cost, $g(n)$
- Online (Path) Cost, g(goal) vs. Offline (Search) Cost
- Heuristic: Estimate of Remaining Path Cost, $h(n)$
- Uniform-Cost (*aka* Branch-and-Bound): $g(n)$ only, $h(n) = 0$

Summary Points

- Reading for Next Class: Sections 3.5 – 3.7, 4.1 – 4.2, R&N 2e
- This Week: Search, Chapters 3 - 4
  - State spaces
  - Graph search examples
  - Basic search frameworks: discrete and continuous
- Uninformed ("Blind") vs. Informed ("Heuristic") Search
  - h(n) and g(n) defined: no h in blind search; online cost = g(goal)
  - Properties: completeness, time and space complexity, offline cost
  - Uniform-cost search (B&B) as generalization of BFS: g(n) only
- Relation to Intelligent Systems Concepts
  - Knowledge representation: evaluation functions, macros
  - Planning, reasoning, learning
- Coming Week: Heuristic Search, Chapter 4
- Later: Goal-Directed Reasoning, Planning (Chapter 11)