Lecture 3 of 42

Search Problems
Discussion: Term Projects 3 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 3.2 – 3.4, p. 62 - 81, Russell & Norvig 2nd edition

Lecture Outline

- Reading for Next Class: Sections 3.2 – 3.4, R&N 2e
- This Week: Search, Chapters 3 - 4
  - State spaces
  - Graph search examples
  - Basic search frameworks: discrete and continuous
- Uninformed ("Blind") and Informed ("Heuristic") Search
  - Cost functions: online vs. offline
  - Time and space complexity
  - Heuristics: examples from graph search, constraint satisfaction
- Relation to Intelligent Systems Concepts
  - Knowledge representation: evaluation functions, macros
  - Planning, reasoning, learning
- Next Week: Heuristic Search, Chapter 4; Constraints, Chapter 5

Term Project Topics (Review)

- **1. Game-playing Expert System**
  - "Borg" for Angband computer role-playing game (CRPG)
  - http://www.thangorodrim.net/borg.html
- **2. Trading Agent Competition (TAC)**
  - Supply Chain Management (SCM) and Classic scenarios
  - http://www.sics.se/tac/page.php?id=13
- **3. Knowledge Base for Bioinformatics**
  - Evidence ontology for genomics or proteomics
  - http://bioinformatics.ai.sri.com/evidence-ontology/

Review: Criteria for Search Strategies

- **Completeness**
  - *Is strategy guaranteed to find solution when one exists?*
  - Typical requirements/assumptions for guaranteed solution
    - Finite depth solution
    - Finite branch factor
    - Minimum unit cost (if paths can be infinite) – discussion: why?
- **Time Complexity**
  - *How long does it take to find solution in worst case?*
  - Asymptotic analysis
- **Space Complexity**
  - *How much memory does it take to perform search in worst case?*
  - Analysis based on data structure used to maintain frontier
- **Optimality**
  - *Finds highest-quality solution when more than one exists?*
  - Quality: defined in terms of node depth, path cost

Review: Uninformed (Blind) Search Strategies

- Breadth-First Search (BFS)
  - Basic algorithm: breadth-first traversal of search tree
  - Intuitive idea: expand whole frontier first
  - Advantages: finds optimal (minimum-depth) solution for finite search spaces
  - Disadvantages: intractable (exponential complexity, high constants)
- Depth-First Search (DFS)
  - Basic algorithm: depth-first traversal of search tree
  - Intuitive idea: expand *one* path first and backtrack
  - Advantages: narrow frontier
  - Disadvantages: lot of backtracking in worst case; suboptimal and incomplete
- Search Issues
  - Criteria: completeness (convergence); optimality; time, space complexity
  - "Blind"
    - No information about number of steps or path cost from state *to goal*
    - *i.e.*, no path cost estimator function (heuristic)
- Uniform-Cost, Depth-Limited, Iterative Deepening, Bidirectional

Review: BFS Algorithm

- function *Breadth-First-Search* (*problem*) returns a solution or failure
  - return *General-Search* (*problem*, *Enqueue-At-End*)
- function *Enqueue-At-End* (*e*: Element-Set) returns void
  - // Queue: priority queue data structure
  - while not (*e.Is-Empty*())
    - if not *queue.Is-Empty*() then *queue.last.next* ← *e.head*();
    - *queue.last* ← *e.head*();
    - *e.Pop-Element*();
  - return
- Implementation Details
  - Recall: *Enqueue-At-End* downward funarg for *Insert* argument of *General-Search*
  - Methods of *Queue* (priority queue)
    - *Make-Queue* (*Element-Set*) – constructor
    - *Is-Empty()* – boolean-valued method
    - *Remove-Front()* – element-valued method
    - *Insert(Element-Set)* – procedure, *aka Queuing-Fn*

Review: BFS Analysis

- **Asymptotic Analysis: Worst-Case Time Complexity**
  - Branching factor: b (max number of children per expanded node)
  - 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: $\Sigma_i b^i = 1 + b + b^2 + \ldots + b^d = O(b^d)$
- **Worst-Case Space Complexity:** $O(b^d)$
- **Properties**
  - Convergence: suppose b, d finite
    - Complete: guaranteed to find *a* solution
    - Optimal: guaranteed to find minimum-depth solution (why?)
  - Very poor worst-case time complexity (see Figure 3.12, R&N)

Uniform-Cost Search [1]: A Generalization of BFS

- **Generalizing to Blind, Cost-Based Search**
- **Justification**
  - BFS: finds shallowest (min-depth) goal state
  - Not necessarily min-cost goal state for general $g(n)$
  - Want: ability to find least-cost solution
- **Modification to BFS**
  - Expand lowest-cost node on fringe
  - Requires *Insert* function to insert into increasing order
  - Alternative conceptualization: *Remove-Front* as *Select-Next*
  - See: Winston, Nilsson
- **BFS: Specific Case of Uniform-Cost Search**
  - $g(n) = depth(n)$
  - In BFS case, optimality guaranteed (discussion: why?)

Uniform-Cost Search [2]: Example

- R&N 2e
- Requirement for Uniform-Cost Search to Find Min-Cost Solution
  - Monotone restriction:

    $g(Successor(n)) \equiv g(n) + cost\ (n, Successor(n)) \geq g(n)$
  - Intuitive idea
    - Cost increases monotonically with search depth (distance from root)
    - i.e., nonnegative edge costs
  - Discussion
    - Always realistic, i.e., can always be expected in real-world situations?
    - What happens if monotone restriction is violated?

Depth-First Search [1]: Algorithm

- **function** *Depth-First-Search* (*problem*) **returns** a solution **or** failure
  - **return** *General-Search* (*problem, Enqueue-At-Front*)
- **function** *Enqueue-At-Front* (*e*: Element-Set) **returns** void
  - // Queue: priority queue data structure
  - **while** not (*e.Is-Empty*())
    - *temp* ← *queue.first*;
    - *queue.first* ← *e.head*();
    - *queue.first.next* ← *temp*;
    - *e.Pop-Element*();
  - **return**
- Implementation Details
  - *Enqueue-At-Front* downward funarg for *Insert* argument of *General-Search*
  - Otherwise similar in implementation to BFS
  - Exercise (easy)
    - Recursive implementation
    - See Cormen, Leiserson, Rivest, & Stein (2002)

Depth-First Search [2]: Analysis

- **Asymptotic Analysis: Worst-Case Time Complexity**
  - Branching factor: b (maximum number of children per expanded node)
  - Max depth (in levels from root, i.e., edge depth): m
  - Analysis
    - b^i nodes generated at level i
    - At least this many nodes to test
    - Total: $\Sigma_i b^i = 1 + b + b^2 + \ldots + b^m = O(b^m)$
- **Worst-Case Space Complexity:** $O(bm)$ – Why?
- **Example: Figure 3.14, R&N**
- **Properties**
  - Convergence: suppose b, m finite
    - Not complete: not guaranteed to find *a* solution (discussion – why?)
    - Not optimal: not guaranteed to find minimum-depth solution
  - Poor worst-case time complexity

Depth-Limited Search: A Bounded Specialization of DFS

- **Intuitive Idea**
  - Impose cutoff on maximum depth of path
  - Search no further in tree
- **Analysis**
  - Max search depth (in levels from root, i.e., edge depth): l
  - Analysis
    - b^i nodes generated at level i
    - At least this many nodes to test
    - Total: $\Sigma_i b^i = 1 + b + b^2 + \ldots + b^l = O(b^l)$
- **Worst-Case Space Complexity:** $O(bl)$
- **Properties**
  - Convergence: suppose b, l finite *and* $l \geq d$
    - Complete: guaranteed to find *a* solution
    - Not optimal: not guaranteed to find minimum-depth solution
  - Worst-case time complexity depends on l, actual solution depth d

Iterative Deepening Search: An Incremental Specialization of DFS

- **Intuitive Idea**
  - Search incrementally
  - Anytime algorithm: return value on demand
- **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: $\Sigma_i b^i = 1 + b + b^2 + \ldots + b^d = O(b^d)$
- **Worst-Case Space Complexity:** $O(bd)$
- **Properties**
  - Convergence: suppose b, l finite *and* $l \geq d$
    - Complete: guaranteed to find *a* solution
    - Optimal: guaranteed to find minimum-depth solution (why?)

Bidirectional Search: A Concurrent Variant of BFS

- **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

Comparison of Search Strategies: Blind / Uninformed Search

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

Informed (Heuristic) Search: Overview

- **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 $\rightarrow$ R; h: node $\rightarrow$ R; f: node $\rightarrow$ R
  - Using *h*
    - ⇨ *h* only: greedy (*aka* myopic) informed search
    - ⇨ $f = g + h$: (some) hill-climbing, A/A*
- **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 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

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?*

CIS 530 / 730
Artificial Intelligence
Lecture 3 of 42
Computing & Information Sciences
Kansas State University

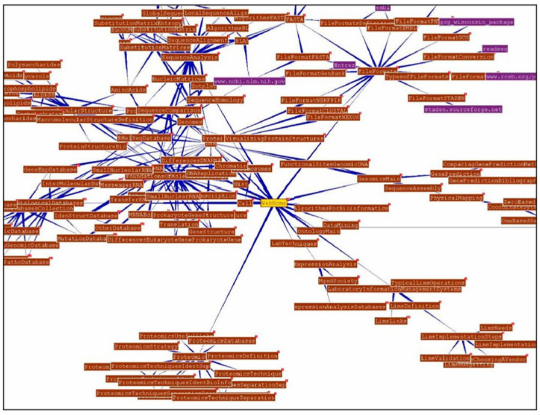
Heuristic Search [2]: Background

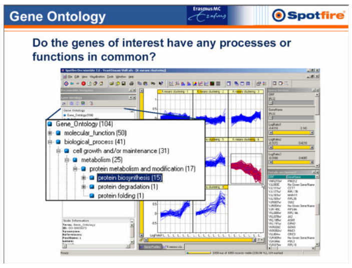
- **Origins of Term**
  - ***Heuriskein*** – to find (to discover)
  - ***Heureka*** ("I have found it") – attributed to Archimedes
- **Usage of Term**
  - Mathematical logic in problem solving
    - ⇨ Polyà [1957]
    - ⇨ Methods for discovering, inventing problem-solving techniques
    - ⇨ Mathematical 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: Overview

- **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?

Example: Data Mining [1]
Project Topic 3 of 5





http://gchelpdesk.ualberta.ca/WebTextBook/CBHDWebTextBookTofC.htm

http://www.erasmusmc.nl/bioinformatics/research/gocp.shtml

Example: Data Mining [2]
Problem Specification

- Data Mining Applications
  - Bioinformatics
  - Social networks
  - Text analytics and visualization (Topic 4, covered in Lecture 4)
- Bioinformatics
  - Develop, convert knowledge base for genomics or proteomics
    http://bioinformatics.ai.sri.com/evidence-ontology/
  - Use ontology development tool
    - PowerLoom: http://www.isi.edu/isd/LOOM/PowerLoom/
    - Semantic Web: http://www.w3.org/TR/owl-ref/
  - Build an ontology for query answering (QA)
  - Test with other ontology reasoners: TAMBIS, semantic web-based
  - Export to / interconvert among languages: Ontolingua, etc.
- Social Networks: Link Analysis
- Text Analytics: More in Natural Language Processing (NLP)

Terminology

- State Space Search
- Goal-Directed Reasoning, Planning
- Search Types: Uninformed ("Blind") vs. Informed ("Heuristic")
- Basic Search Algorithms
  - British Museum (depth-first *aka* DFS), iterative-deepening DFS
  - Breadth-First *aka* BFS, depth-limited, uniform-cost
  - Bidirectional
  - Branch-and-Bound
- Properties of Search
  - Soundness: returned candidate path satisfies specification
  - Completeness: finds path if one exists
  - Optimality: (usually means) achieves maximal online path cost
  - Optimal efficiency: (usually means) maximal offline cost

Summary Points

- Reading for Next Class: Sections 3.2 – 3.4, R&N 2e
- This Week: Search, Chapters 3 - 4
  - State spaces
  - Graph search examples
  - Basic search frameworks: discrete and continuous
- Uninformed ("Blind") and Informed ("Heuristic") Search
  - Cost functions: online vs. offline
  - Time and space complexity
  - Heuristics: examples from graph search, constraint satisfaction
- Relation to Intelligent Systems Concepts
  - Knowledge representation: evaluation functions, macros
  - Planning, reasoning, learning
- Next Week: Heuristic Search, Chapter 4; Constraints, Chapter 5
- Later: Goal-Directed Reasoning, Planning (Chapter 11)