



Lecture 3 of 42

Search Problems Discussion: Problem Set 1, Term Projects 3 of 3

Monday, 27 August 2007

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/Fall-2007/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 2^e
- 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, Fall 2007 (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 (TAC-SCM) scenario
 - * <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/>



Search and Problem-Solving Agents: Review

- **Problem Solving in Agent Frameworks**
 - * Recall: intelligent agent (IA) scenario from Chapter 2, R&N
 - * Last week: design, spec, implementation of IA “skeleton”
- **Cost Measures in Search**
 - * Actual vs. Heuristic (estimated)
 - * Offline (computational) vs. Online (path)
- **Kinds of Search**
 - * **Costs used**
 - ⇒ Actual, actual + heuristic
 - ⇒ Offline + online
 - * **Uninformed (blind) search: today and Wednesday**
 - ⇒ Sometimes cost-based
 - ⇒ Often exhaustive
 - * **Informed (heuristic): Friday and next week**
 - ⇒ Minimize actual + heuristic cost
 - ⇒ Can still guarantee optimal online cost? Offline cost?





General Search [1]: Overview

- **Generating Action Sequences**
 - * **Initialization:** start (initial) state
 - * **Test for goal condition**
 - ⇒ Membership in goal state set (explicitly enumerated)
 - ⇒ Constraints met (implicit)
 - * **Applying operators (when goal state not achieved)**
 - ⇒ Implementation: generate new set of successor (child) states
 - ⇒ Conceptual process: expand state
 - ⇒ Result: *multiple branches* (e.g., Figure 3.8 R&N)
- **Intuitive Idea**
 - * **Select one option**
 - * **Ordering** (prioritizing / scheduling) others for later consideration
 - * **Iteration:** choose, test, expand
 - * **Termination:** solution is found or no states remain to be expanded
- **Search Strategy:** Selection of State to Be Expanded



General Search [2]: Algorithm

- **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*



Search Strategies: Criteria

- **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



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**





Breadth-First Search [1]: 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*



Breadth-First Search [2]: 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: $\sum_i b^i = 1 + b + b^2 + \dots + b^d = O(b^d)$
- **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
 - * $g(n) = \text{depth}(n)$
 - * 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:

$$g(\text{Successor}(n)) \equiv g(n) + \text{cost}(n, \text{Successor}(n)) \geq g(n)$$
 - * 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: $\sum_i b^i = 1 + b + b^2 + \dots + b^m = O(b^m)$
- **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: $\sum_i b^i = 1 + b + b^2 + \dots + b^l = O(b^l)$
- **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: $\sum_i b^i = 1 + b + b^2 + \dots + b^d = O(b^d)$
- **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: $\sum_i b^i = 1 + b + b^2 + \dots + b^{d/2} = O(b^{d/2})$
- **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





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 $\rightarrow \mathbb{R}$; 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*
- **Branch and Bound Search**
 - * Originates from operations research (OR)
 - * Special case of heuristic search: treat as $h(n) = 0$
 - * Sort candidates by $g(n)$



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*: $\text{node} \rightarrow \mathbb{R}$
 - ⇒ *Queueing-Fn* \equiv *Sort-By*: $\text{node list} \rightarrow \text{node list}$
 - * 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?





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?*



Next Topic: Informed (Heuristic) 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**

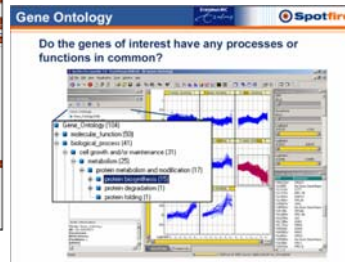
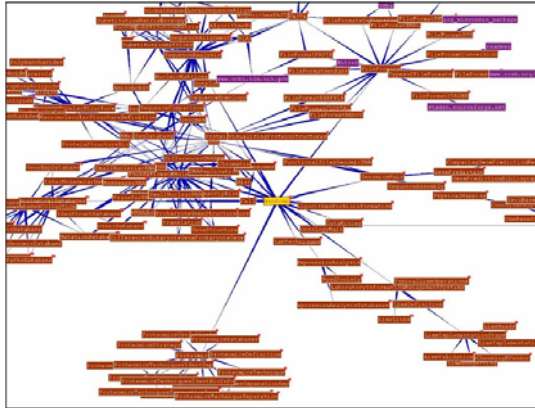
● **This Week: More Search Basics**

- * Memory bounded, iterative improvement (gradient, Monte Carlo search)
- * Introduction to game tree search





Example: Bioinformatics KB [1] Project Topic 3 of 3



© 2005 Adam Shlien
<http://gchelpdesk.ualberta.ca/WebTextBook/CBHDWebTextBookTofC.htm>
 © 2002 Gene Ontology Consortium
<http://www.erasmusmc.nl/bioinformatics/research/gocp.shtml>



Example: Bioinformatics KB [2] Problem Specification

- **Knowledge Base for Bioinformatics**
 - * Evidence ontology for genomics or proteomics
 - * <http://bioinformatics.ai.sri.com/evidence-ontology/>
- **Problem Specification**
 - * Use ontology development tools
 - ⇒ PowerLoom: <http://www.isi.edu/isd/LOOM/PowerLoom/>
 - ⇒ Semantic Web: <http://www.w3.org/TR/owl-ref/>
 - * Study and convert existing bioinformatics knowledge bases
 - * Build an ontology for query answering (QA)
 - ⇒ Goal: improve QA capability using available knowledge
 - ⇒ Technical requirements: interface to ontology reasoner, new ontology
 - * Test with other ontology reasoners
 - ⇒ TAMBIS (Stevens *et al.*)
 - ⇒ Semantic web-based (OWL)
 - * Export to / interconvert among languages: Ontolingua, etc.





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 2^e**
- **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)**