



Lecture 4 of 42

Uninformed Search: DLS, Bidirectional, B&B, SMA, IDA

Wednesday, 30 August 2006

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-2006/CIS730>

Instructor home page: <http://www.cis.ksu.edu/~bhsu>

Reading for Next Class:

Sections 3.5 – 3.7, p. 81 – 88, Russell & Norvig 2nd edition

Section 4.1, p. 91 - 105, Russell & Norvig 2nd edition



Lecture Outline

- Reading for Next Class: Sections 3.5 – 3.7, 4.1, R&N 2^e
- This Week: Search, Chapters 3 - 4
 - * State spaces
 - * Graph search examples
 - * Basic search frameworks: discrete and continuous
- Coping with Time and Space Limitations of Uninformed Search
 - * Depth-limited and memory-bounded search
 - * Iterative deepening
 - * Bidirectional search
- Intro to Heuristic Search
 - * What is a heuristic?
 - * Relationship to optimization, static evaluation, bias in learning
 - * Desired properties and applications of heuristics
- Friday, Monday: Heuristic Search, Chapter 4





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*



Iterative Deepening Search: Review

- **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: 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: $\sum_i b^i = 1 + b + b^2 + \dots + b^{d/2} = O(b^{d/2})$
- **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



Informed (Heuristic) Search: Review

- **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*: node → 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?



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?*



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 → 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?



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**



Greedy Search [1]: A Best-First Algorithm

- **function *Greedy-Search (problem)* returns solution or failure**
 - * // recall: solution Option
 - * return *Best-First-Search (problem, h)*
- **Example of Straight-Line Distance (SLD) Heuristic: Figure 4.2 R&N**
 - * **Can only calculate if city locations (coordinates) are known**
 - * **Discussion: Why is h_{SLD} useful?**
 - ⇒ Underestimate
 - ⇒ Close estimate
- **Example: Figure 4.3 R&N**
 - * **Is solution optimal?**
 - * **Why or why not?**





Greedy Search [2]: Properties

- **Similar to DFS**
 - * Prefers single path to goal
 - * Backtracks
- **Same Drawbacks as DFS?**
 - * Not optimal
 - ⇒ First solution
 - ⇒ Not necessarily best
 - ⇒ Discussion: How is this problem mitigated by quality of h ?
 - * Not complete: doesn't consider cumulative cost "so-far" (g)
- **Worst-Case Time Complexity: $O(b^m)$ – Why?**
- **Worst-Case Space Complexity: $O(b^m)$ – Why?**



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





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)**