



## Lecture 5 of 42

### Informed Search Intro

Friday, 31 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 4.2 – 4.3, p. 105 – 116, Russell & Norvig 2<sup>nd</sup> edition



## Lecture Outline

- **Reading for Next Class: Sections 4.2 – 4.3, R&N 2<sup>e</sup>**
- **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
- **Next Week: Heuristic Search, Constraints, Intro to Games**



| $d$ | Search Cost |            |            | Effective Branching Factor |            |            |
|-----|-------------|------------|------------|----------------------------|------------|------------|
|     | IDS         | $A^*(h_1)$ | $A^*(h_2)$ | IDS                        | $A^*(h_1)$ | $A^*(h_2)$ |
| 2   | 10          | 6          | 6          | 2.45                       | 1.79       | 1.79       |
| 4   | 112         | 13         | 12         | 2.87                       | 1.48       | 1.45       |
| 6   | 680         | 20         | 18         | 2.73                       | 1.34       | 1.30       |
| 8   | 6384        | 39         | 25         | 2.80                       | 1.33       | 1.24       |
| 10  | 47127       | 93         | 39         | 2.79                       | 1.38       | 1.22       |
| 12  | 364435      | 227        | 73         | 2.78                       | 1.42       | 1.24       |
| 14  | –           | 539        | 113        | –                          | 1.44       | 1.23       |
| 16  | –           | 1301       | 211        | –                          | 1.45       | 1.25       |
| 18  | –           | 3056       | 363        | –                          | 1.46       | 1.26       |
| 20  | –           | 7276       | 676        | –                          | 1.47       | 1.27       |
| 22  | –           | 18094      | 1219       | –                          | 1.48       | 1.28       |
| 24  | –           | 39135      | 1641       | –                          | 1.48       | 1.26       |

**Figure 4.8** Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and  $A^*$  algorithms with  $h_1$ ,  $h_2$ . Data are averaged over 100 instances of the 8-puzzle, for various solution lengths.

Inventing admissible heuristic functions



## Review: Best-First Search [1]

- **Evaluation Function**
  - \* Recall: *General-Search* (Figure 3.9, 3.10 R&N)
  - \* **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: Family of Algorithms**
  - \* **Justification:** using only  $g$  doesn't *direct search toward goal*
  - \* Nodes ordered so 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”**
  - \* “Apparent best node based on eval function applied to current frontier”
  - \* **Discussion:** when is best-first not really best?



## Review: Best-First Search [2]

- **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?*



## Heuristic Search [2]: Background

- **Origins of Term**
  - \* *Heuriskein* – to find (to discover)
  - \* *Heureka*
    - ⇒ “I have found it”
    - ⇒ Legend imputes exclamation to Archimedes (bathtub flotation / displacement)
- **Usage of Term**
  - \* Mathematical logic in problem solving
    - ⇒ Polya [1957]
    - ⇒ Study of methods for discovering and 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)**
  - \* Any standard (symbolic rule or quantitative measure) used to *reduce search*
  - \* “As opposed to exhaustive blind search”



## 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?



## Greedy Search [4]: More Properties

- Good Heuristic Functions Reduce Practical Space and Time Complexity
  - \* "Your mileage may vary": actual reduction
    - ⇒ Domain-specific
    - ⇒ Depends on quality of  $h$  (what quality  $h$  can we achieve?)
  - \* "You get what you pay for": computational costs or knowledge required
- Discussions and Questions to Think About
  - \* How much is search reduced using straight-line distance heuristic?
  - \* When do we prefer analytical vs. search-based solutions?
  - \* What is the complexity of an exact solution?
  - \* Can "meta-heuristics" be derived that meet our desiderata?
    - ⇒ Underestimate
    - ⇒ Close estimate
  - \* When is it feasible to develop parametric heuristics automatically?
    - ⇒ Finding underestimates
    - ⇒ Discovering close estimates





## Algorithm A/A\* [1]: Methodology

- Idea: Combine Evaluation Functions  $g$  and  $h$ 
  - \* Get “best of both worlds”
  - \* Discussion: Why is it important to take both components into account?
- function  $A$ -Search ( $problem$ ) returns solution or failure
  - \* // recall: solution Option
  - \* return  $Best\text{-}First\text{-}Search(problem, g + h)$
- Requirement: Monotone Restriction on  $f$ 
  - \* Recall: monotonicity of  $h$ 
    - ⇒ Requirement for completeness of uniform-cost search
    - ⇒ Generalize to  $f = g + h$
  - \* aka triangle inequality
- Requirement for  $A = A^*$ : Admissibility of  $h$ 
  - \*  $h$  must be an underestimate of the true optimal cost ( $\forall n . h(n) \leq h^*(n)$ )



## Algorithm A/A\* [2]: Properties

- Completeness (p. 100 R&N)
  - \* Expand lowest-cost node on fringe
  - \* Requires  $Insert$  function to insert into increasing order
- Optimality (p. 99-101 R&N)
- Optimal Efficiency (p. 97-99 R&N)
  - \* For any given heuristic function
  - \* No other optimal algorithm is guaranteed to expand fewer nodes
  - \* Proof sketch: by contradiction (on what partial correctness condition?)
- Worst-Case Time Complexity (p. 100-101 R&N)
  - \* Still exponential in solution length
  - \* Practical consideration: *optimally efficient* for any given heuristic function





## Algorithm A/A\* [3]:

### Optimality/Completeness and Performance

- Admissibility: Requirement for A\* Search to Find Min-Cost Solution
- Related Property: Monotone Restriction on Heuristics
  - \* For all nodes  $m, n$  such that  $m$  is a descendant of  $n$ :  $h(m) \geq h(n) - c(n, m)$
  - \* Change in  $h$  is less than true cost
  - \* Intuitive idea: "No node looks artificially distant from a goal"
  - \* Discussion questions
    - ⇒ Admissibility  $\Rightarrow$  monotonicity? Monotonicity  $\Rightarrow$  admissibility?
    - ⇒ Always realistic, i.e., can always be expected in real-world situations?
    - ⇒ What happens if monotone restriction is violated? (Can we fix it?)
- Optimality and Completeness
  - \* Necessarily and sufficient condition (NASC): admissibility of  $h$
  - \* Proof: p. 99-100 R&N (contradiction from inequalities)
- Behavior of A\*: Optimal Efficiency
- Empirical Performance
  - \* Depends very much on how tight  $h$  is

CIS 530 / 7.00: Artificial Intelligence: How might this problem arise? Friday, 31 Aug 2017

Computing & Information Sciences  
Kansas State University



## Problems with Best-First Searches

- Idea: Optimization-Based Problem Solving as Function Maximization
  - \* Visualize function space: criterion (z axis) versus solutions (x-y plane)
  - \* Objective: maximize criterion subject to solutions, degrees of freedom
- Foothills aka Local Optima
  - \* aka relative minima (of error), relative maxima (of criterion)
  - \* Qualitative description
    - ⇒ All applicable operators produce suboptimal results (i.e., neighbors)
    - ⇒ However, solution is not optimal!
  - \* Discussion: Why does this happen in optimization?
- Lack of Gradient aka Plateaux
  - \* Qualitative description: all neighbors indistinguishable by evaluation function  $f$
  - \* Related problem: jump discontinuities in function space
  - \* Discussion: When does this happen in heuristic problem solving?
- Single-Step Traps aka Ridges
  - \* Qualitative description: unable to move along steepest gradient

CIS 530 / 7.00: Artificial Intelligence: How might this problem arise? Friday, 31 Aug 2017

Computing & Information Sciences  
Kansas State University



## Heuristic Functions

- **Examples**
  - \* Euclidean distance
  - \* Combining heuristics
    - ⇒ Evaluation *vector* → evaluation *matrix*
    - ⇒ Combining *functions*: minimization, more sophisticated combinations
- **Performance**
  - \* Theory
    - ⇒ Admissible  $h$  ⇒ existence of monotonic  $h$  (pathmax heuristic)
    - ⇒ Admissibility ⇒ optimal with algorithm A (i.e., A\*)
    - ⇒ A\* is optimally efficient for any heuristic
  - \* Practice: admissible heuristic could still be bad!
- **Developing Heuristics Automatically: “Solve the Right Problem”**
  - \* Relaxation methods
    - ⇒ Solve an easier problem
    - ⇒ Dynamic programming in graphs: known shortest-paths to “nearby” states
  - \* Feature extraction



## Preview: Iterative Improvement Framework

- **Intuitive Idea**
  - \* “Single-point search frontier”
    - ⇒ Expand one node at a time
    - ⇒ Place children at head of queue
    - ⇒ *Sort only this sublist, by  $f$*
  - \* Result – direct convergence in direction of steepest:
    - ⇒ Ascent (in criterion)
    - ⇒ Descent (in error)
  - \* Common property: proceed toward goal *from search locus (or loci)*
- **Variations**
  - \* Local (steepest ascent hill-climbing) versus global (simulated annealing)
  - \* Deterministic versus Monte-Carlo
  - \* Single-point versus multi-point
    - ⇒ Maintain frontier
    - ⇒ Systematic search (cf. OPEN / CLOSED lists): parallel simulated annealing
    - ⇒ Search with recombination: genetic algorithm



## Preview: Hill-Climbing (Gradient Descent)

- function *Hill-Climbing* (*problem*) returns solution state
  - \* inputs: *problem*: specification of problem (structure or class)
  - \* static: *current*, *next*: search nodes
  - \*  $current \leftarrow \text{Make-Node}(\text{problem.Initial-State})$
  - \* loop do
    - $\Rightarrow next \leftarrow$  a highest-valued successor of *current*
    - $\Rightarrow$  if *next.value()* < *current.value()* then return *current*
    - $\Rightarrow current \leftarrow next$  // make transition
  - \* end
- Steepest Ascent Hill-Climbing
  - \* aka gradient ascent (descent)
  - \* Analogy: finding “tangent plane to objective surface”
  - \* Implementations
    - $\Rightarrow$  Finding derivative of (differentiable) *f* with respect to parameters
    - $\Rightarrow$  Example: error backpropagation in artificial neural networks (later)



## Search-Based Problem Solving: Quick Review

- function *General-Search* (*problem*, *strategy*) returns a solution or failure
  - \* Queue: represents search frontier (see: Nilsson – OPEN / CLOSED lists)
  - \* Variants: based on “add resulting nodes to search tree”
- Previous Topics
  - \* Formulating *problem*
  - \* Uninformed search
    - $\Rightarrow$  No heuristics: only  $g(n)$ , if any cost function used
    - $\Rightarrow$  Variants: BFS (uniform-cost, bidirectional), DFS (depth-limited, ID-DFS)
  - \* Heuristic search
    - $\Rightarrow$  Based on  $h$  – (heuristic) function, returns estimate of min cost to goal
    - $\Rightarrow$   $h$  only: greedy (aka myopic) informed search
    - $\Rightarrow$  A/A\*:  $f(n) = g(n) + h(n)$  – frontier based on *estimated* + *accumulated* cost
- Today: More Heuristic Search Algorithms
  - \* A\* extensions: iterative deepening (IDA\*) and simplified memory-bounded (SMA\*)





## Properties of Algorithm A/A\*: Review

- Admissibility: Requirement for A\* Search to Find Min-Cost Solution
- Related Property: Monotone Restriction on Heuristics
  - \* For all nodes  $m, n$  such that  $m$  is a descendant of  $n$ :  $h(m) \geq h(n) - c(n, m)$
  - \* Discussion questions
    - ⇒ Admissibility ⇒ monotonicity? Monotonicity ⇒ admissibility?
    - ⇒ What happens if monotone restriction is violated? (Can we fix it?)
- Optimality Proof for Admissible Heuristics
  - \* Theorem: If  $\forall n . h(n) \leq h^*(n)$ , A\* will never return a suboptimal goal node.
  - \* Proof
    - ⇒ Suppose A\* returns  $x$  such that  $\exists s . g(s) < g(x)$
    - ⇒ Let path from root to  $s$  be  $\langle n_0, n_1, \dots, n_k \rangle$  where  $n_k \equiv s$
    - ⇒ Suppose A\* expands a subpath  $\langle n_0, n_1, \dots, n_j \rangle$  of this path
    - ⇒ Lemma: by induction on  $i$ ,  $s = n_k$  is expanded as well
      - Base case:  $n_0$  (root) always expanded
      - Induction step:  $h(n_{i+1}) \leq h^*(n_{i+1})$ , so  $f(n_{i+1}) \leq f(x)$ , Q.E.D.
    - ⇒ Contradiction: if  $s$  were expanded, A\* would have selected  $s$ , not  $x$



## A/A\*: Extensions (IDA\*, SMA\*)

- Memory-Bounded Search
  - \* Rationale
    - ⇒ Some problems intrinsically difficult (intractable, exponentially complex)
    - ⇒ Fig. 3.12, p. 75 R&N (compare Garey and Johnson, Baase, Sedgewick)
    - ⇒ "Something's got to give" – size, time or memory? ("Usually it's memory")
- Iterative Deepening A\* – Pearl, Rorf (Fig. 4.10, p. 107 R&N)
  - \* Idea: use iterative deepening DFS with sort on  $f$  – expands node iff A\* does
  - \* Limit on expansion:  $f$ -cost
  - \* Space complexity: linear in depth of goal node
  - \* Caveat: could take  $O(n^2)$  time – e.g., TSP ( $n = 10^6$  could still be a problem)
  - \* Possible fix
    - ⇒ Increase  $f$  cost limit by  $\epsilon$  on each iteration
    - ⇒ Approximation error bound: no worse than  $\epsilon$ -bad ( $\epsilon$ -admissible)
- Simplified Memory-Bounded A\* – Chakrabarti, Russell (Fig. 4.12 p. 107 R&N)
  - \* Idea: make space on queue as needed (compare: virtual memory)





## Iterative Improvement: Framework

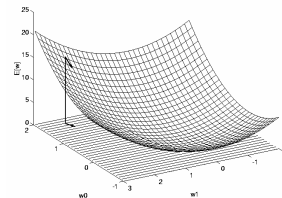
- **Intuitive Idea**
  - \* “Single-point search frontier”
    - ⇒ Expand one node at a time
    - ⇒ Place children at head of queue
    - ⇒ *Sort only this sublist, by f*
  - \* **Result** – direct convergence in direction of steepest:
    - ⇒ Ascent (in criterion)
    - ⇒ Descent (in error)
  - \* Common property: proceed toward goal *from search locus (or loci)*
- **Variations**
  - \* Local (steepest ascent hill-climbing) versus global (simulated annealing)
  - \* Deterministic versus Monte-Carlo
  - \* Single-point versus multi-point
    - ⇒ Maintain frontier
    - ⇒ Systematic search (cf. OPEN / CLOSED lists): parallel simulated annealing
    - ⇒ Search with recombination: genetic algorithm



## Hill-Climbing [1]: An Iterative Improvement Algorithm

- function *Hill-Climbing (problem)* returns solution state
  - \* inputs: *problem*: specification of problem (structure or class)
  - \* static: *current, next*: search nodes
  - \* *current* ← *Make-Node (problem.Initial-State)*
  - \* loop do
    - ⇒ *next* ← a highest-valued successor of *current*
    - ⇒ if *next.value()* < *current.value()* then return *current*
    - ⇒ *current* ← *next* // make transition
  - \* end
- **Steepest Ascent Hill-Climbing**
  - \* aka gradient ascent (descent)
  - \* Analogy: finding “tangent plane to objective surface”
  - \* Implementations
    - ⇒ Finding derivative of (differentiable) *f* with respect to parameters
    - ⇒ Example: error backpropagation in artificial neural networks (later)

$$\nabla E[\vec{w}] \equiv \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$





## Terminology

- **Heuristic Search Algorithms**
  - \* Properties of *heuristics*: monotonicity, admissibility
  - \* Properties of *algorithms*: completeness, optimality, optimal efficiency
  - \* Iterative improvement
    - ⇒ Hill-climbing
    - ⇒ Beam search
    - ⇒ Simulated annealing (SA)
  - \* Function maximization formulation of search
  - \* **Problems**
    - ⇒ Ridge
    - ⇒ Foothill aka local (relative) optimum aka local minimum (of error)
    - ⇒ Plateau, jump discontinuity
  - \* **Solutions**
    - ⇒ Macro operators
    - ⇒ Global optimization (genetic algorithms / SA)
- **Constraint Satisfaction Search**



## Summary Points

- **More Heuristic Search**
  - \* **Best-First Search: A/A\*** concluded
  - \* **Iterative improvement**
    - ⇒ Hill-climbing
    - ⇒ Simulated annealing (SA)
  - \* **Search as function maximization**
    - ⇒ **Problems:** ridge; foothill; plateau, jump discontinuity
    - ⇒ **Solutions:** macro operators; global optimization (genetic algorithms / SA)
- **Next Lecture: AI Applications 1 of 3**
- **Next Week: Adversarial Search (e.g., Game Tree Search)**
  - \* **Competitive problems**
  - \* **Minimax algorithm**