A/A*, Beam Search, and Iterative Improvement

Wednesday, 01 September 2004

William H. Hsu
Department of Computing and Information Sciences, KSU
http://www.kddresearch.org
http://www.cis.ksu.edu/~bhsu

Reading for Next Week:
Handouts #1-2 (Nilsson, Ginsberg) – at Fiedler Hall Copy Center
Chapter 6, Russell and Norvig
• Today’s Reading
  – Sections 4.3 – 4.5, Russell and Norvig
  – Recommended references: Chapter 4, Ginsberg; Chapter 3, Winston
• Reading for Next Week: Chapter 6, Russell and Norvig 2e
• 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
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 → R; \( h \): node → R; \( f \): node → 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]: Characterization of Algorithm Family

• 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 that 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 [2]: 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”
    • Legend imputes exclamation to Archimedes (bathtub flotation / displacement)

• Usage of Term
  – Mathematical logic in problem solving
    • Polyà [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”
  – 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?
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 $Best$-$First$-$Search\ (problem, g + h)$

- **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
  - **How weak is admissibility as a practical requirement?**
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
  – **Discussion**: *How might this problem be overcome?*
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
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 versu**s **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 ← 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)

- **Discussion**: Difference Between Hill-Climbing, Best-First?

CIS 730: Introduction to Artificial Intelligence
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
    - No heuristics: only $g(n)$, if any cost function used
    - Variants: BFS (uniform-cost, bidirectional), DFS (depth-limited, ID-DFS)
  - Heuristic search
    - Based on $h$ – (heuristic) function, returns estimate of min cost to goal
    - $h$ only: greedy (*aka* myopic) informed search
    - $A/A^*$: $f(n) = g(n) + h(n)$ – frontier based on estimated + accumulated cost
- **Today: More Heuristic Search Algorithms**
  - $A^*$ extensions: iterative deepening (IDA*) and simplified memory-bounded (SMA*)
  - Iterative improvement: hill-climbing, MCMC (simulated annealing)
  - Problems and solutions (macros and global optimization)
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 \( < n_0, n_1, ..., n_k > \) where \( n_k = s \)
    - Suppose \( A^* \) expands a subpath \( < n_0, n_1, ..., n_j > \) of this path
    - **Lemma**: by induction on \( i \), \( s = n_k \) is expanded as well
      - **Base case**: \( n_0 \) (root) always expanded
      - **Induction step**: \( h(n_{j+1}) \leq h^*(n_{j+1}) \), so \( f(n_{j+1}) \leq f(x) \), 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)
  - Selective forgetting: drop nodes (select victims) with highest $f$
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)
- **Discussion:** Difference Between Hill-Climbing, Best-First?
Hill-Climbing [2]:
A Restriction of Best-First Search

- **Discussion**: How is Hill-Climbing a Restriction of Best-First?
- **Answer**: Dropped Condition
  - **Best first**: sort by $h$ or $f$ over *current frontier*
    - Compare: insert each element of expanded node into queue, in order
    - Result: greedy search ($h$) or A/A* ($f$)
  - **Hill climbing**: sort by $h$ or $f$ within *child list of current node*
    - Compare: local bucket sort
    - **Discussion (important)**: Does it matter whether we include $g$?

- **Impact of Modification on Algorithm**
  - Search time complexity decreases
  - Comparison with A/A* (Best-First using $f$)
    - *Still optimal?* No
    - *Still complete?* Yes
  - Variations on hill-climbing (later): momentum, random restarts
Hill-Climbing [3]: Local Optima (Foothill Problem)

- **Local Optima** *aka* Local Trap States
- **Problem Definition**
  - Point reached by hill-climbing may be maximal but not maximum
  - **Maximal**
    - **Definition**: not dominated by any neighboring point *(with respect to criterion measure)*
    - In this partial ordering, maxima are incomparable
  - **Maximum**
    - **Definition**: dominates all neighboring points *(wrt criterion measure)*
    - Different partial ordering imposed: “z value”
- **Ramifications**
  - Steepest ascent hill-climbing will become trapped *(why?)*
  - Need some way to break out of trap state
    - Accept transition (i.e., search move) to dominated neighbor
    - Start over: random restarts
Hill-Climbing [4]:
Lack of Gradient (Plateau Problem)

- **Zero Gradient Neighborhoods** *aka* Plateaux
- **Problem Definition**
  - Function space may contain points whose neighbors are *indistinguishable* *(wrt* criterion measure)*
  - Effect: “flat” search landscape
  - **Discussion**
    - *When does this happen in practice?*
    - *Specifically, for what kind of heuristics might this happen?*
- **Ramifications**
  - Steepest ascent hill-climbing will become trapped *(why?)*
  - Need some way to break out of zero gradient
    - Accept transition (i.e., search move) to random neighbor
    - Random restarts
    - *Take bigger steps* *(later, in planning)*
Hill-Climbing [5]:
Single-Step Traps (Ridge Problem)

- **Single-Step Traps** *aka* **Ridges**

- **Problem Definition**
  - Function space may contain points such that single move in any “direction” leads to suboptimal neighbor
  - **Effect**
    - There exists steepest gradient to goal
    - None of allowed steps moves along that gradient
    - Thin “knife edge” in search landscape, hard to navigate
    - **Discussion (important):** *When does this occur in practice?*
      - *NB:* ridges can lead to local optima, too

- **Ramifications**
  - Steepest ascent hill-climbing will become trapped (*why?*)
  - Need some way to break out of ridge-walking
    - **Formulate** composite transition (multi-dimension step) – *how?*
    - **Accept** multi-step transition (at least one to worse state) – *how?*
    - Random restarts
Ridge Problem Solution: Multi-Step Trajectories (Macros)

- Intuitive Idea: Take More than One Step in Moving along Ridge
- Analogy: Tacking in Sailing
  - Need to move against wind direction
  - Have to compose move from multiple small steps
    - Combined move: in (or more toward) direction of steepest gradient
    - Another view: decompose problem into self-contained subproblems
- Multi-Step Trajectories: Macro Operators
  - Macros: (inductively) generalize from 2 to > 2 steps
  - Example: Rubik’s Cube
    - Can solve 3 x 3 x 3 cube by solving, interchanging 2 x 2 x 2 cubies
    - Knowledge used to formulate subcube (cubie) as macro operator
      - Treat operator as single step (multiple primitive steps)
- Discussion: Issues
  - How can we be sure macro is atomic? What are pre-, postconditions?
  - What is good granularity (length in primitives) for macro in our problem?
Plateau, Local Optimum, Ridge Solution: Global Optimization

• Intuitive Idea
  – Allow search algorithm to take some “bad” steps to escape from trap states
  – Decrease probability of taking such steps gradually to prevent return to traps

• Analogy: Marble(s) on Rubber Sheet
  – Goal: move marble(s) into global minimum from any starting position
  – Shake system: hard at first, gradually decreasing vibration
  – Marbles tend to break out of local minima but have less chance of re-entering

• Analogy: Annealing
  – Ideas from metallurgy, statistical thermodynamics
  – Cooling molten substance: slow as opposed to rapid (quenching)
  – Goal: maximize material strength of solidified substance (e.g., metal or glass)

• Multi-Step Trajectories in Global Optimization: Super-Transitions

• Discussion: Issues
  – How can we be sure macro is atomic? What are pre-, postconditions?
  – What is good granularity (length in primitives) for macro in our problem?
Beam Search: “Parallel” Hill-Climbing

- **Idea**
  - Teams of climbers
    - Communicating by radio
    - Frontier is only \( w \) teams wide (\( w \equiv \text{beam width} \))
    - Expand cf. best-first but take best \( w \) only \textit{per layer}
  - Synchronous search: push frontier forward at uniform depth from start node

- **Algorithm Details**
  - How do we order OPEN (priority queue) by \( h \)?
  - How do we maintain CLOSED?

- **Question**
  - What behavior does beam search with \( w = 1 \) exhibit?
  - Hint: only one “team”, can’t split up!
  - Answer: equivalent to hill-climbing

- **Other Properties, Design Issues**
  - Another analogy: flashlight \textit{beam} with adjustable radius (hence name)
  - What should \( w \) be? How will this affect solution quality?
Iterative Improvement

Global Optimization (GO) Algorithms

- Idea: Apply Global Optimization with Iterative Improvement
  - Iterative improvement: local transition (primitive step)
  - Global optimization algorithm
    - “Schedules” exploration of landscape
    - Selects next state to visit
    - Guides search by specifying probability distribution over local transitions

- Brief History of Markov Chain Monte Carlo (MCMC) Family
  - MCMC algorithms first developed in 1940s (Metropolis)
  - First implemented in 1980s
    - “Optimization by simulated annealing” (Kirkpatrick, Gelatt, Vecchi, 1983)
    - Boltzmann machines (Ackley, Hinton, Sejnowski, 1985)
  - Tremendous amount of research and application since
    - Neural, genetic, Bayesian computation
    - See: CIS730 Class Resources page
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