CS:4420 Artificial Intelligence Spring 2018

Informed Search

Cesare Tinelli

The University of Iowa

Copyright 2004–18, Cesare Tinelli and Stuart Russell^{*a*}

^a These notes were originally developed by Stuart Russell and are used with permission. They are copyrighted material and may not be used in other course settings outside of the University of Iowa in their current or modified form without the express written consent of the copyright holders.

Readings

• Chap. 3 of [Russell and Norvig, 2012]

Review: Tree Search

function TREE-SEARCH(problem, fringe) returns a solution, or failure fringe \leftarrow INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe) loop do if fringe is empty then return failure $node \leftarrow$ REMOVE-FRONT(fringe) if GOAL-TEST[problem] applied to STATE(node) succeeds return node fringe \leftarrow INSERTALL(EXPAND(node, problem), fringe)

A strategy is defined by a particular node expansion order

Uninformed search strategies look for solutions by systematically generating new states and checking each of them against the goal

This approach is very inefficient in most cases

Most successor states are "obviously" a bad choice

Such strategies do not know that because they have minimal problem-specific knowledge

Informed search strategies exploit problem-specific knowledge as much as possible to drive the search

They are almost always more efficient than uninformed searches and often also optimal

Main Idea

- Use the knowledge of the problem domain to build an evaluation function f
- For every node n in the search space, f(n) quantifies the desirability of expanding n in order to reach the goal
- Then use the desirability value of the nodes in the fringe to decide which node to expand next

The evaluation function f is typically an imperfect measure of the goodness of the node

I.e., the right choice of nodes is not always the one suggested by f

The evaluation function f is typically an imperfect measure of the goodness of the node

I.e., the right choice of nodes is not always the one suggested by f

Note: It is possible to build a perfect evaluation function, which will always suggest the right choice

How? Why don't we use perfect evaluation functions then?

Standard Assumptions on Search Spaces

- The cost of a node increases with the node's depth
- Transitions costs are non-negative and bounded below, i.e., there is a $\epsilon > 0$ such that the cost of each transition is $\geq \epsilon$
- Each node has only finitely-many successors

Note: There are problems that do not satisfy one or more of these assumptions

Best-First Search

Idea: use an evaluation function estimating the desirability of each node

Strategy: Always expand the most desirable unexpanded node

Implementation: the *fringe* is a priority queue sorted in decreasing order of desirability

Special cases:

- greedy search
- A* search

Best-First Search

Idea: use an evaluation function estimating the desirability of each node

Strategy: Always expand the most desirable unexpanded node

Implementation: the *fringe* is a priority queue sorted in decreasing order of desirability

Special cases:

- greedy search
- A* search

Note: Since f is only an approximation, "Best-First" is a misnomer. Each time we choose the node at that point appears to be the best

Best-first Search Strategies

Best-first is a family of search strategies, each with a different evaluation function

Typically, strategies use estimates of the cost of reaching the goal and try to minimize it

Uniform Search also tries to minimize a cost measure. Is it then a best-first search strategy?

Best-first Search Strategies

Best-first is a family of search strategies, each with a different evaluation function

Typically, strategies use estimates of the cost of reaching the goal and try to minimize it

Uniform Search also tries to minimize a cost measure. Is it then a best-first search strategy?

Not in spirit, because the evaluation function should incorporate a cost estimate of going from the current state to the closest goal state

Romania with Step Costs in Km



Greedy Best-First Search

Evaluation function h(n) (heuristics)

 $= \quad \text{estimate cost of cheapest path} \\ \text{from node } n \text{ to closest goal}$

E.g., $h_{SLD}(n) = \text{straight-line distance from } n$ to Bucharest

Greedy search expands the node that appears to be closest to goal









Complete?

Time complexity?

Space complexity?

Optimal?

CS:4420 Spring 2018 - p.13/31

Complete? Only in finite spaces with repeated-state checking Otherwise, can get stuck in loops: Iasi \rightarrow Neamt \rightarrow Iasi \rightarrow Neamt \rightarrow

Time complexity?

Space complexity?

Time complexity? $O(b^m)$ — may have to expand all nodes

Space complexity?

Complete? Only in finite spaces with repeated-state checking Otherwise, can get stuck in loops: Iasi \rightarrow Neamt \rightarrow Iasi \rightarrow Neamt \rightarrow Time complexity? $O(b^m)$ — may have to expand all nodes Space complexity? $O(b^m)$ — may have to keep most nodes in memory

Complete? Only in finite spaces with repeated-state checking Otherwise, can get stuck in loops: Iasi \rightarrow Neamt \rightarrow Iasi \rightarrow Neamt \rightarrow Time complexity? $O(b^m)$ — may have to expand all nodes Space complexity? $O(b^m)$ — may have to keep most nodes in memory

Optimal? No

Complete? Only in finite spaces with repeated-state checking Otherwise, can get stuck in loops: Iasi \rightarrow Neamt \rightarrow Iasi \rightarrow Neamt \rightarrow Time complexity? $O(b^m)$ — may have to expand all nodes Space complexity? $O(b^m)$ — may have to keep most nodes

in memory

Optimal? No

A good heuristic can nonetheless produce dramatic time/space improvements in practice

A* — A Better Best-First Strategy

Greedy Best-first search

- minimizes estimated cost h(n) from current node n to goal
- is informed but almost always suboptimal and incomplete

Uniform cost search

- minimizes actual cost g(n) to current node n
- is, in most cases, optimal and complete but uninformed

A* search

- combines the two by minimizing f(n) = g(n) + h(n)
- is, *under reasonable assumptions*, optimal and complete, and also informed

Idea: avoid expanding paths that are already expensive

Evaluation function: f(n) = g(n) + h(n)

g(n) = cost so far to reach n h(n) = estimated cost to goal from nf(n) = estimated total cost of path through n to goal

A* search should use an *admissible* heuristic:

for all n, $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n

E.g., $h_{SLD}(n)$ never overestimates the actual road distance













A* Search: Why an Admissible Heuristic

If h is admissible, f(n) never overestimates the actual cost of the best solution through n

Overestimates are dangerous



Consistent heuristics

A heuristic is *consistent* if

 $h(n) \le c(n, a, n') + h(n')$

If f is consistent, we have

$$f(n') = g(n') + h(n')$$

= $g(n) + c(n, a, n') + h(n')$
$$\geq g(n) + h(n)$$

= $f(n)$



I.e., f(n) is nondecreasing along any path

Consistent heuristics

A heuristic is *consistent* if

$$h(n) \le c(n, a, n') + h(n')$$

If f is consistent, we have

$$f(n') = g(n') + h(n')$$

= $g(n) + c(n, a, n') + h(n')$
$$\geq g(n) + h(n)$$

= $f(n)$



I.e., f(n) is nondecreasing along any path

Note:

- Consistent \Rightarrow admissible
- Most admissible heuristics are also consistent

Let h be any admissible heuristic function

Theorem: A^* search using h is optimal

Proof is easier under the stronger assumption that h is consistent

Let h be any admissible heuristic function

Theorem: A^* search using h is optimal

Proof is easier under the stronger assumption that h is consistent

A* expands some nodes with $f(n) = C^*$ A* expands no nodes with $f(n) > C^*$

A* expands all nodes with $f(n) < C^* = \text{cost of optimal goal}$

Let h be any admissible heuristic function

Theorem: A^* search using h is optimal

Proof is easier under the stronger assumption that h is consistent

 $\begin{array}{ll} \mathsf{A}^* \text{ expands all nodes with} & f(n) < C^* = \text{cost of optimal goal} \\ \mathsf{A}^* \text{ expands some nodes with} & f(n) = C^* \\ \mathsf{A}^* \text{ expands no nodes with} & f(n) > C^* \end{array}$

Theorem: A* is *optimally efficient* for h: no other optimal strategy using h expands fewer nodes than A*

Optimality of A*: Basic Argument

Suppose some suboptimal goal G_2 has been generated and is in the queue. Let n be an unexpanded node on a least-cost path to an optimal goal G



$$f(G_2) = g(G_2) \qquad \text{since } h(G_2) = 0$$

$$> g(G) \qquad \text{since } G_2 \text{ is suboptimal}$$

$$\ge f(n) \qquad \text{since } h \text{ is admissible}$$

Since $f(G_2) > f(n)$, A* will never select G_2 for expansion

Optimality of A* (more useful)

Lemma: A* expands nodes in order of increasing f-value It gradually adds "f-contours" of nodes (cf. breadth-first adds layers) Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



- Complete?
- Time complexity?
- Space complexity?
- Optimal?

Complete? Yes, unless there are infinitely many nodes n with $f(n) \leq f(G)$

Time complexity?

Space complexity?

Complete? Yes, unless there are infinitely many nodes n with $f(n) \leq f(G)$

$$\begin{array}{lll} \epsilon = & \left| h(n_0) - h^*(n_0) \right| \\ \mbox{Time complexity?} & O(b^{\epsilon d}) \mbox{ where } & n_0 = & \mbox{start state} \\ & h^* = & \mbox{actual cost to goal state} \end{array}$$

Space complexity?

Complete? Yes, unless there are infinitely many nodes n with $f(n) \leq f(G)$

 $\begin{array}{lll} \epsilon = & \left| h(n_0) - h^*(n_0) \right| \\ \mbox{Time complexity?} & O(b^{\epsilon d}) \mbox{ where } & n_0 = & \mbox{start state} \\ & h^* = & \mbox{actual cost to goal state} \end{array}$

Subexponential only in uncommon case where $\epsilon \leq O(\log h^*(n_0))$

Space complexity?

Complete? Yes, unless there are infinitely many nodes n with $f(n) \leq f(G)$

 $\begin{array}{lll} \epsilon = & \left| h(n_0) - h^*(n_0) \right| \\ \mbox{Time complexity?} & O(b^{\epsilon d}) \mbox{ where } & n_0 = & \mbox{start state} \\ & h^* = & \mbox{actual cost to goal state} \end{array}$

Subexponential only in uncommon case where $\epsilon \leq O(\log h^*(n_0))$

Space complexity? $O(b^m)$, as in Greedy Best-First — may end up with all nodes in memory

Complete? Yes, unless there are infinitely many nodes n with $f(n) \leq f(G)$

 $\begin{array}{lll} \epsilon = & \left| h(n_0) - h^*(n_0) \right| \\ \mbox{Time complexity?} & O(b^{\epsilon d}) \mbox{ where } & n_0 = & \mbox{start state} \\ & h^* = & \mbox{actual cost to goal state} \end{array}$

Subexponential only in uncommon case where $\epsilon \leq O(\log h^*(n_0))$

Space complexity? $O(b^m)$, as in Greedy Best-First — may end up with all nodes in memory

Optimal? Yes if h is admissible (and standard assumptions hold) — cannot expand f_{i+1} until f_i is finished

Beyond A*

A* generally runs out of memory before it runs out of time

Other best-first strategies keep the good properties on A* while trying to reduce memory consumption:

- Recursive Best-First search (RBFS)
- Iterative Deepening A* (IDA*)
- Memory-bounded A* (MA*)
- Simple Memory-bounded A* (SMA*)

Admissible Heuristics

A* search is optimal with an admissible heuristic function h

How do we devise good heuristic functions for a given problem?

Typically, that depends on the problem domain

However, there are some general techniques that work reasonably well across several domains

Examples of Admissible Heuristics

8-puzzle problem:

- $h_1(n) =$ number of tiles in the wrong position at state n
- $h_2(n) = \text{sum of the Manhattan distances of each tile from its goal position}$



Start State



Goal State

- $h_1(Start) =$
- $h_2(Start) =$

Examples of Admissible Heuristics

8-puzzle problem:

- $h_1(n) =$ number of tiles in the wrong position at state n
- $h_2(n) = \text{sum of the Manhattan distances of each tile from its goal position}$



Start State



Goal State

- $h_1(Start) = 7$
- $h_2(Start) =$

Examples of Admissible Heuristics

8-puzzle problem:

- $h_1(n) =$ number of tiles in the wrong position at state n
- $h_2(n) = \text{sum of the Manhattan distances of each tile from its goal position}$



Start State



Goal State

- $h_1(Start) = 7$
- $h_2(Start) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14$

Dominance

A heuristic function h_2 dominates a heuristic function h_1 for a problem P if $h_2(n) \ge h_1(n)$ for all nodes n in P's space

Ex.: the 8-puzzle

 $h_2 = total Manhattan distance$ dominates $h_1 = number of misplaced tiles$

With A*, if h_2 is admissible and dominates h_1 , then it is always better for search: A* will never expand more nodes with h_2 than with h_1

What if neither of h_1, h_2 dominates the other? If both h_1, h_2 are admissible, use $h(n) = \max(h_1(n), h_2(n))$

Effectiveness of Heuristic Functions

Let

- h be a heuristic function h for A*
- N the total number of nodes expanded by one A* search with h
- d the depth of the found solution

The effective branching Factor (EBF) of h is the value b^* that solves the equation

$$x^{d} + x^{d-1} + \dots + x^{2} + x + 1 - N = 0$$

(the branching factor of a uniform tree with N nodes and depth d)

A heuristics h for A^* is effective in practice if its average EBF is close to 1

Note: If h_2 dominates h_2 , then $\mathsf{EFB}(h_2) \leq \mathsf{EFB}(h_1)$

Dominance and EFB: The 8-puzzle

	Search Cost			Effective Branching Factor		
d	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	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	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

Average values over 1200 random instances of the problem

- d depth of solution
- Search cost # of expanded nodes
- IDS iterative deepening search
- h_1 number of misplaces tiles
- h_2 total Manhattan distance

Devising Heuristic Functions

A relaxed problem is a version of a search problem with less restrictions on the applicability of the next-state operators

Example: *n*-puzzle

- original: "A tile can move from position p to position q if p is adjacent to q and q is empty"
- relaxed-1: "A tile can move from p to q if p is adjacent to q"
- relaxed-2: "A tile can move from p to q if q is empty"
- relaxed-3: "A tile can move from p to q"

The exact solution cost of a relaxed problem is often a good (admissible) heuristics for the original problem

Key point: the optimal solution cost of the relaxed problem is no greater than the optimal solution cost of the original problem

Relaxed Problems: Another Example

Traveling salesperson problem

Original problem: Find the shortest tour visiting n cities exactly once Complexity: NP-complete

Relaxed problem: Find a tree with the smallest cost that connects the n cities (minimum spanning tree)

Complexity: $O(n^2)$

Cost of tree is a lower bound on the shortest (open) tour



CS:4420 Spring 2018 - p.30/31

Devising Heuristic Functions Automatically

- Relaxation of formally described problems
- Pattern databases
- Learning