Set 3: Informed Heuristic Search

ICS 271 Fall 2013
Kalev Kask
Overview

• Heuristics and Optimal search strategies
  – heuristics
  – hill-climbing algorithms
  – Best-First search
  – A*: optimal search using heuristics
  – Properties of A*
    • admissibility,
    • consistency,
    • accuracy and dominance
    • Optimal efficiency of A*
  – Branch and Bound
  – Iterative deepening A*
  – Automatic generation of heuristics
Heuristic Search

• State-Space Search: every problem is like search of a map
• A problem solving robot finds a path in a state-space graph from start state to goal state, using heuristics

Heuristic = straight-line distance
State Space for Path Finding in a Map
State Space for Path Finding in a Map
Greedy Search Example
8-puzzle: 181,440 states
15-puzzle: 1.3 trillion
24-puzzle: $10^{25}$

Search space exponential

Use Heuristics as people do
State Space of the 8 Puzzle Problem

$h_1 = \text{number of misplaced tiles}$

$h_2 = \text{Manhattan distance}$

Figure 3.6 State space of the 8-puzzle generated by “move blank” operations.
What are Heuristics

- Rule of thumb, intuition
- A quick way to estimate how close we are to the goal. How close is a state to the goal.
- Pearl: “the ever-amazing observation of how much people can accomplish with that simplistic, unreliable information source known as intuition.”

8-puzzle

- $h_1(n)$: number of misplaced tiles
- $h_2(n)$: Manhattan distance

\[
\begin{align*}
 h_1(S) &= ? \quad 8 \\
 h_2(S) &= ? \quad 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18
\end{align*}
\]

- Path-finding on a map
  - Euclidean distance
Problem: Finding a Minimum Cost Path

• Previously we wanted an arbitrary path to a goal or best cost. Now, we want the minimum cost path to a goal G
  – Cost of a path = sum of individual transitions along path

• Examples of path-cost:
  – Navigation
    • path-cost = distance to node in miles
      – minimum => minimum time, least fuel
  – VLSI Design
    • path-cost = length of wires between chips
      – minimum => least clock/signal delay
  – 8-Puzzle
    • path-cost = number of pieces moved
      – minimum => least time to solve the puzzle

• Algorithm: Uniform-cost search... still somewhat blind
Heuristic Functions

- **8-puzzle**
  - Number of misplaced tiles
  - Manhatten distance
  - Gaschnig’s

- **8-queen**
  - Number of future feasible slots
  - Min number of feasible slots in a row
  - Min number of conflicts (in complete assignments states)

- **Travelling salesperson**
  - Minimum spanning tree
  - Minimum assignment problem
Best-First (Greedy) Search:

$$f(n) = \text{number of misplaced tiles}$$

Figure 8.1

Start and Goal Configurations for the Eight-Puzzle
Romania with Step Costs in km
Greedy Best-First Search

• Evaluation function $f(n) = h(n)$ (heuristic)
  
  = estimate of cost from $n$ to goal

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

• Greedy best-first search expands the node that appears to be closest to goal
Greedy Best-First Search Example
Greedy Best-First Search Example
Greedy Best-First Search Example
Greedy Best-First Search Example
Problems with Greedy Search

• Not complete
• Get stuck on local minima and plateaus,
• Irrevocable,
• Infinite loops
• Can we incorporate heuristics in systematic search?
Informed Search - Heuristic Search

• How to use heuristic knowledge in systematic search?
• Where? (in node expansion? hill-climbing?)
• Best-first:
  – select the best from **all** the nodes encountered so far in OPEN.
  – “good” use heuristics
• Heuristic estimates value of a node
  – promise of a node
  – difficulty of solving the subproblem
  – quality of solution represented by node
  – the amount of information gained.
• f(n)- heuristic evaluation function.
  – depends on n, goal, search so far, domain
A* Search

- Idea: avoid expanding paths that are already expensive

- Evaluation function $f(n) = g(n) + h(n)$

- $g(n) = \text{cost so far to reach } n$

- $h(n) = \text{estimated cost from } n \text{ to goal}$

- $f(n) = \text{estimated total cost of path through } n \text{ to goal}$
A* Search Example
A* Search Example
A* Search Example
A* Search Example
A* Search Example
A* Search Example
A*- a Special Best-First Search

• Goal: find a minimum sum-cost path
• Notation:
  – $c(n,n')$ - cost of arc $(n,n')$
  – $g(n) = \text{cost of current path from start to node } n \text{ in the search tree.}$
  – $h(n) = \text{estimate of the cheapest cost of a path from } n \text{ to a goal.}$
  – Special evaluation function: $f = g + h$
• $f(n)$ estimates the cheapest cost solution path that goes through $n$.
  – $h^*(n)$ is the true cheapest cost from $n$ to a goal.
  – $g^*(n)$ is the true shortest path from the start $s$, to $n$.

• If the heuristic function, $h$ always underestimate the true cost ($h(n)$ is smaller than $h^*(n)$), then A* is guaranteed to find an optimal solution.
A* on 8-Puzzle with $h(n) = \# \text{ misplaced tiles}$
Algorithm A* (with any h on search Graph)

- Input: an implicit search graph problem with cost on the arcs
- Output: the minimal cost path from start node to a goal node.
  - 1. Put the start node s on OPEN.
  - 2. If OPEN is empty, exit with failure
  - 3. Remove from OPEN and place on CLOSED a node n having minimum f.
  - 4. If n is a goal node exit successfully with a solution path obtained by tracing back the pointers from n to s.
  - 5. Otherwise, expand n generating its children and directing pointers from each child node to n.
    - For every child node n’ do
      - evaluate h(n’) and compute f(n’) = g(n’) + h(n’) = g(n) + c(n, n’) + h(n’)
      - If n’ is already on OPEN or CLOSED compare its new f with the old f. If the new value is higher, discard the node.
      - Else, put n’ with its f value in the right order in OPEN
  - 6. Go to step 2.
Best-First Algorithm $BF$ (*)

1. Put the start node $s$ on a list called $OPEN$ of unexpanded nodes.
2. If $OPEN$ is empty exit with failure; no solutions exists.
3. Remove the first $OPEN$ node $n$ at which $f$ is minimum (break ties arbitrarily), and place it on a list called $CLOSED$ to be used for expanded nodes.
4. Expand node $n$, generating all it’s successors with pointers back to $n$.
5. If any of $n$’s successors is a goal node, exit successfully with the solution obtained by tracing the path along the pointers from the goal back to $s$.
6. For every successor $n'$ on $n$:
   a. Calculate $f(n')$.
   b. if $n'$ was neither on $OPEN$ nor on $CLOSED$, add it to $OPEN$. Attach a pointer from $n'$ back to $n$. Assign the newly computed $f(n')$ to node $n'$.
   c. if $n'$ already resided on $OPEN$ or $CLOSED$, compare the newly computed $f(n')$ with the value previously assigned to $n'$. If the old value is lower, discard the newly generated node. If the new value is lower, substitute it for the old ($n'$ now points back to $n$ instead of to its previous predecessor). If the matching node $n'$ resided on $CLOSED$, move it back to $OPEN$.
7. Go to step 2.

* With tests for duplicate nodes.
271-Fall 2013
Example of A* Algorithm in Action
Behavior of A - Termination

• The heuristic function \( h(n) \) is called admissible if \( h(n) \) is never larger than \( h^*(n) \), namely \( h(n) \) is always less or equal to true cheapest cost from \( n \) to the goal.

• \( A^* \) is admissible if it uses an admissible heuristic, and \( h(\text{goal}) = 0 \).

• Theorem (completeness) (Hart, Nilsson and Raphael, 1968)
  
  – \( A^* \) always terminates with a solution path (\( h \) is not necessarily admissible) if
    • costs on arcs are positive, above epsilon
    • branching degree is finite.

• Proof: The evaluation function \( f \) of nodes expanded must increase eventually (since paths are longer and more costly) until all the nodes on an optimal path are expanded.
Behavior of A* - Completeness

• Theorem (completeness for optimal solution) (HNL, 1968):
  – If the heuristic function is admissible than A* finds an optimal solution.

• Proof:
  – 1. A* will expand only nodes whose f-values are less (or equal) to the optimal cost path C* (f(n) is less-or-equal C*).
  – 2. The evaluation function of a goal node along an optimal path equals C*.

• Lemma:
  – Anytime before A* terminates there exists and OPEN node n’ on an optimal path with f(n’) <= C*.
Consistent (monotone) Heuristics

- A heuristic is **consistent** if for every node \( n \), every successor \( n' \) of \( n \) generated by any action \( a \),

\[
h(n) \leq c(n,a,n') + h(n')
\]

- If \( h \) 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 non-decreasing along any path.

- **Theorem**: If \( h(n) \) is consistent, \( f \) along any path is non-decreasing.
- **Corollary**: the \( f \) values seen by A* are non-decreasing.
Consistent Heuristics

- If $h$ is consistent and $h(\text{goal})=0$ then $h$ is admissible
  - Proof: (by induction of distance from the goal)

- An A* guided by consistent heuristic finds an optimal paths to all expanded nodes, namely $g(n) = g^*(n)$ for any closed $n$.
  - Proof: Assume $g(n) > g^*(n)$ and $n$ expanded along a non-optimal path.
    - Let $n'$ be the shallowest OPEN node on optimal path $p$ to $n$ →
    - $g(n') = g^*(n')$ and therefore $f(n')=g^*(n')+h(n')$
    - Due to consistency we get $f(n') \leq g^*(n')+k(n',n)+h(n)$
    - Since $g^*(n) = g^*(n')+k(n',n)$ along the optimal path, we get that
      - $f(n') \leq g^*(n) + h(n)$
    - And since $g(n) > g^*(n)$ then $f(n') < g(n)+h(n) = f(n)$, contradiction
A* with Consistent Heuristics

- A* expands nodes in order of increasing $f$ value
- Gradually adds "$f$-contours" of nodes
- Contour $i$ has all nodes with $f=f_i$ where $f_i < f_{i+1}$
Summary of Consistent Heuristics

• h is consistent if the heuristic function satisfies triangle inequality for every n and its child node n’: h(ni) ≤ h(nj) + c(ni,nj)

• When h is consistent, the f values of nodes expanded by A* are never decreasing.
• When A* selected n for expansion it already found the shortest path to it.
• When h is consistent every node is expanded once (if check for duplicates).
• Normally the heuristics we encounter are consistent
  – the number of misplaced tiles
  – Manhattan distance
  – straight-line distance
Admissible and Consistent Heuristics?

E.g., for the 8-puzzle:

- $h_1(n) = \text{number of misplaced tiles}$
- $h_2(n) = \text{total Manhattan distance}$
  (i.e., no. of squares from desired location of each tile)

The true cost is 26.
Average cost for 8-puzzle is 22. Branching degree 3.

- $h_1(S) = \ ? = 8$
- $h_2(S) = \ ? = 3+1+2+2+2+3+3+2 = 18$
Summary so far

• Heuristic (informed) search
  – Best-First (guided by heuristic eval fn f)
  – A* as a special case of BF (f=g+h)
  – A* is guaranteed to terminate as long as costs>0 and branching factor is finite (any h)
  – A* is guaranteed to find optimal if h is admissible
    • If not admissible, still useful, but optimality not guaranteed
  – Consistent (monotonic) h
    • Consistent implies admissible
    • When a node is selected for expansion, shortest path to it is found
    • More efficient, do not have to re-open expanded (closed) nodes
    • Expands nodes in increasing order of f
A* properties

• A* expands every path along which $f(n) < C^*$

• A* will never expand any node s.t. $f(n) > C^*$

• If $h$ is consistent A* will expand any node such that $f(n) < C^*$

• Therefore, A* expands all the nodes for which $f(n) < C^*$ and a subset of the nodes for which $f(n) = C^*$.

• Therefore, if $h_1(n) < h_2(n)$ clearly the subset of nodes expanded by $h_2$ is smaller.
Complexity of A*

• A* is optimally efficient (Dechter and Pearl 1985):
  – It can be shown that all algorithms that do not expand a node which A* did expand (inside the contours) may miss an optimal solution

• A* worst-case time complexity:
  – is exponential unless the heuristic function is very accurate

• If $h$ is exact ($h = h^*$)
  – search focus only on optimal paths

• Main problem: space complexity is exponential

• Effective branching factor:
  – logarithm of base $(d+1)$ of average number of nodes expanded.
Effectiveness of A* search

• How quality of heuristic impact search?

• What is the time and space complexity?

• Is any algorithm better? Worse?

• Case study: the 8-puzzle
## Effectiveness of A* Search Algorithm

### Average number of nodes expanded

<table>
<thead>
<tr>
<th>d</th>
<th>IDS</th>
<th>A*(h1)</th>
<th>A*(h2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>6384</td>
<td>39</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>364404</td>
<td>227</td>
<td>73</td>
</tr>
<tr>
<td>14</td>
<td>3473941</td>
<td>539</td>
<td>113</td>
</tr>
<tr>
<td>20</td>
<td>------------</td>
<td>7276</td>
<td>676</td>
</tr>
<tr>
<td>24</td>
<td>------------</td>
<td>39135</td>
<td>1641</td>
</tr>
</tbody>
</table>

Average over 100 randomly generated 8-puzzle problems

- $h_1$ = number of tiles in the wrong position
- $h_2$ = sum of Manhattan distances

271-Fall 2013
Dominance

• Definition: If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible) then $h_2$ dominates $h_1$

• Is $h_2$ better for search?

• Typical search costs (average number of nodes expanded):

  • $d=12$  
    IDS = 3,644,035 nodes  
    $A^*(h_1) = 227$ nodes  
    $A^*(h_2) = 73$ nodes

  • $d=24$  
    IDS = out of memory  
    $A^*(h_1) = 39,135$ nodes  
    $A^*(h_2) = 1,641$ nodes
Heuristic’s Dominance and Pruning Power

• Definition:
  – A heuristic function $h_2$ (strictly) dominates $h_1$ if both are admissible and for every node $n$, $h_2(n)$ is (strictly) greater than $h_1(n)$.

• Theorem (Hart, Nilsson and Raphael, 1968):
  – An A* search with a dominating heuristic function $h_2$ has the property that any node it expands is also expanded by A* with $h_1$.

• Question: Does Manhattan distance dominate the number of misplaced tiles?

• Extreme cases
  – $h = 0$
  – $h = h^*$
A* with Consistent Heuristics

- Contours of stronger (dominating) heuristics are inside contours of weaker heuristics
Properties of A*

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

Time?? Exponential in [relative error in $h \times$ length of soln.]

Space?? Keeps all nodes in memory

Optimal?? Yes—cannot expand $f_{i+1}$ until $f_i$ is finished

A* expands all nodes with $f(n) < C^*$

A* expands some nodes with $f(n) = C^*$

A* expands no nodes with $f(n) > C^*$
Relationships among Search Algorithms

Depth first
(LIFO ordering)

\( \hat{f} = \text{depth} \)
(Breadth first)

\( \hat{h} = 0 \)
(Uniform cost)

\( \hat{h} \leq h \)

A*

\( \hat{f} = \hat{g} + \hat{h} \)
(Best-first search)

(Generic graph-search algorithms)
Example of Branch and Bound in action
Pseudocode for Branch and Bound Search
(An informed depth-first search)

Initialize: Let Q = {S}, L=∞

While Q is not empty
    pull Q1, the first element in Q
    if f(Q1)≥L, skip it
    if Q1 is a goal compute the cost of the solution and update
        L ←← minimum (new cost, old cost)
    else
        child_nodes = expand(Q1),
        <eliminate child_nodes which represent simple loops>,
        For each child node n do:
            evaluate f(n). If f(n) is greater than L discard n.
        end-for
        Put remaining child_nodes on top of queue in the order of their f.
    end

Continue
Properties of Branch-and-Bound

• Not guaranteed to terminate unless
  – has depth-bound
  – consistent f and reasonable L

• Optimal:
  – finds an optimal solution (f is admissible)

• Time complexity: exponential

• Space complexity: can be linear

• Advantage:
  – anytime property

• Note: unlike A*, BnB may (will) expand nodes f>C*.
Iterative Deepening A* (IDA*)
(combining Branch-and-Bound and A*)

• Initialize: $f \leftarrow$ the evaluation function of the start node
• until goal node is found
  – Loop:
    • Do Branch-and-bound with upper-bound $L$ equal to current evaluation function $f$.
    • Increment evaluation function to next contour level
  – end

• Properties:
  – Guarantee to find an optimal solution
  – time: exponential, like A*
  – space: linear, like B&B.

  – Problems: The number of iterations may be large.
Inventing Heuristics automatically

- **Examples of Heuristic Functions for A***
  - The 8-puzzle problem
    - The number of tiles in the wrong position
      - is this admissible?
    - Manhattan distance
      - is this admissible?

- How can we invent admissible heuristics in general?
  - look at “relaxed” problem where constraints are removed
    - e.g., we can move in straight lines between cities
    - e.g., we can move tiles independently of each other
Inventing Heuristics Automatically (cont.)

• How did we
  – find h1 and h2 for the 8-puzzle?
  – verify admissibility?
  – prove that straight-line distance is admissible? MST admissible?
• Hypothetical answer:
  – Heuristic are generated from relaxed problems
  – Hypothesis: relaxed problems are easier to solve
• In relaxed models the search space has more operators or more directed arcs
• Example: 8 puzzle:
  – Rule : a tile can be moved from A to B, iff
    • A and B are adjacent
    • B is blank
  – We can generate relaxed problems by removing one or more of the conditions
    • ... A and B are adjacent & B is blank
    • ... if B is blank
Relaxed Problems

• A problem with fewer restrictions on the actions is called a relaxed problem.

• The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

• If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ (number of misplaced tiles) gives the shortest solution.

• If the rules are relaxed so that a tile can move to any $h/v$ adjacent square, then $h_2(n)$ (Manhatten distance) gives the shortest solution.
Generating heuristics (cont.)

• Example: TSP

• Find a tour. A tour is:
  – 1. A graph with subset of edges
  – 2. Connected
  – 3. Total length of edges minimized
  – 4. Each node has degree 2

• Eliminating 4 yields MST.
Relaxed problems contd.

Well-known example: travelling salesperson problem (TSP)
Find the shortest tour visiting all cities exactly once

Minimum spanning tree can be computed in $O(n^2)$
and is a lower bound on the shortest (open) tour
Automating Heuristic generation

• Use STRIPs language representation:
  • Operators:
    – pre-conditions, add-list, delete list
• 8-puzzle example:
  – on(x,y), clear(y) adj(y,z), tiles x1,...,x8
• States: conjunction of predicates:
  – on(x1,c1), on(x2,c2)....on(x8,c8), clear(c9)
• move(x,c1,c2) (move tile x from location c1 to location c2)
  – pre-cond: on(x1,c1), clear(c2), adj(c1,c2)
  – add-list: on(x1,c2), clear(c1)
  – delete-list: on(x1,c1), clear(c2)
• Relaxation:
  – Remove from precondition: clear(c2), adj(c2,c3) \rightarrow \#misplaced tiles
  – Remove clear(c2) \rightarrow Manhattan distance
  – Remove adj(c2,c3) \rightarrow h3, a new procedure that transfers to the empty location a tile appearing there in the goal
• The space of relaxations can be enriched by predicate refinements
  – adj(y,z) = iff neighbour(y,z) and same-line(y,z)

271-Fall 2013
Heuristic generation

• Theorem: Heuristics that are generated from relaxed models are consistent.

• Proof: \( h \) is true shortest path in a relaxed model
  – \( h(n) \leq c'(n,n') + h(n') \) (\( c' \) are shortest distances in relaxed graph)
  – \( c'(n,n') \leq c(n,n') \)
  – \( \rightarrow h(n) \leq c(n,n') + h(n') \)
Heuristic generation

• Total (time) complexity = heuristic computation + nodes expanded

• More powerful heuristic – harder to compute, but more pruning power (fewer nodes expanded)

• Problem:
  – not every relaxed problem is easy
    • How to recognize a relaxed easy problem
    • A proposal: a problem is easy if it can be solved optimally by a greedy algorithm

• Q: what if neither $h_1$ nor $h_2$ is clearly better? $\max(h_1, h_2)$

• Often, a simpler problem which is more constrained is easier; will provide a good upper-bound.
Improving Heuristics

- Reinforcement learning.
- Pattern Databases: you can solve optimally a sub-problem
Pattern Databases

- For sliding tiles and Rubic’s cube

- For a subset of the tiles compute shortest path to the goal using breadth-first search

- For 15 puzzles, if we have 7 fringe tiles and one blank, the number of patterns to store are $16!/(16-8)! = 518,918,400$.

- For each table entry we store the shortest number of moves to the goal from the current location.

- Use different subsets of tiles and take the max heuristic during IDA* search. The number of nodes to solve 15 puzzles was reduced by a factor of 346 (Culberson and Schaeffer)

- How can this be generalized? (a possible project)
Problem-reduction representations
AND/OR search spaces

• The erratic vacuum world (actions are non-deterministic)
• Graphical models
• Decomposable production systems (Natural language parsing)
  Initial database: (C,B,Z)
  Rules: R1: C \(\rightarrow\) (D,L)
         R2: C \(\rightarrow\) (B,M)
         R3: B \(\rightarrow\) (M,M)
         R4: Z \(\rightarrow\) (B,B,M)
  Find a path generating a string with M’s only.
• The tower of Hanoi
  To move n disks from peg 1 to peg 3 using peg 2
  Move n-1 pegs to peg 2 via peg 3,
  move the nth disk to peg 3,
  move n-1 disks from peg 2 to peg 3 via peg 1.
AND/OR Graphs

• Nodes represent subproblems
  – AND links represent subproblem decompositions
  – OR links represent alternative solutions
  – Start node is initial problem
  – Terminal nodes are solved subproblems

• Solution graph
  – It is an AND/OR subgraph such that:
    • It contains the start node
    • All its terminal nodes (nodes with no successors) are solved primitive problems
    • If it contains an AND node A, it must contain the entire group of AND links that leads to children of A.
Algorithms searching AND/OR graphs

- All algorithms generalize using hyper-arc successors rather than simple arcs.

- AO*: is A* that searches AND/OR graphs for a solution subgraph.

- The cost of a solution graph is the sum cost of its arcs. It can be defined recursively as: \( k(n,N) = c_n + k(n_1,N) + \ldots + k(n_k,N) \)

- \( h^*(n) \) is the cost of an optimal solution graph from \( n \) to a set of goal nodes.

- \( h(n) \) is an admissible heuristic for \( h^*(n) \).
  - Monotonicity:
    - \( h(n) \leq c + h(n_1) + \ldots + h(n_k) \) where \( n_1, \ldots, n_k \) are successors of \( n \)
  - AO* is guaranteed to find an optimal solution when it terminates if the heuristic function is admissible.
Beyond Classical Search (chapter 4)

• Local search for optimization
  – Greedy hill-climbing search, simulated annealing, local beam search, genetic algorithms.
  – Local search in continuous spaces
  – SLS: "Like climbing Everest in thick fog with amnesia"

• Searching with non-deterministic actions
  – The erratic vacuum cleaner example
  – Using AND/OR search spaces; solution is a contingent plan.

• Searching with partial observations
  – Using belief states

• Online search agents and unknown environments
  – Actions, costs, goal-tests are revealed in state only
  – Exploration problems. Safely explorable
Summary

• In practice we often want the goal with the minimum cost path

• Exhaustive search is impractical except on small problems

• Heuristic estimates of the path cost from a node to the goal can be efficient in reducing the search space.

• The A* algorithm combines all of these ideas with admissible heuristics (which underestimate), guaranteeing optimality.

• Properties of heuristics:
  – admissibility, consistency, dominance, accuracy

• Reading
  – R&N Chapters 3-4

271-Fall 2013