Problem Solving & Heuristic Search

Byoung-Tak Zhang
School of Computer Science and Engineering
Seoul National University
Search

- **Set of states** that we can be in
  - Including an initial state and goal states

- **For every state**, a set of **actions** that we can take
  - Each action results in a new state
  - Typically defined by successor function
    - Given a state, produces all states that can be reached from it

- **Cost function** that determines the cost of each action
  (or path = sequence of actions)

- **Solution**: path from initial state to a goal state
  - Optimal solution: solution with minimal cost
A simple example: traveling on a graph

start state

C -> 2
A
B -> 3
D
E
F
9
4

http://www.cs.duke.edu/courses/fall08/cps270/
Example: 8-puzzle

- states: integer location of tiles
- operators: move blank left, right, up, down
- path cost: 1 per move
- goal test: = goal state (given)

(optimal solution of n-Puzzle family is NP-hard)
Example: 8-puzzle

Example: 8-puzzle

http://www.cs.duke.edu/courses/fall08/cps270/
Search algorithms

• Uninformed Search
  – Breadth-First Search
  – Depth-First Search
  – Uniform-Cost Search

• Informed Search (Heuristic Search)
  – Greedy Search
  – A* Search

• Local Search
  – Hill-climbing Search
  – Simulated Annealing
  – Local Beam Search
  – Genetic Algorithms
Search strategies

• A strategy is defined by picking the order of node expansion

• Strategies are evaluated along the following dimensions:
  – completeness – does it always find a solution if one exists?
  – time complexity – number of nodes generated/expanded
  – space complexity – maximum number of nodes in memory
  – optimality – does it always find a least-cost solution?

• Time and space complexity are measured in terms of
  – $b$ – maximum branching factor of the search tree
  – $d$ – depth of the least-cost solution
  – $m$ – maximum depth of the state space
Uninformed search

• Given a state, we only know whether it is a goal state or not
• Cannot say one non-goal state looks better than another non-goal state
• Can only traverse state space blindly in hope of somehow hitting a goal state at some point
  – Also called blind search
  – Blind does not imply unsystematic
Breadth-first search

Expand shallowest unexpanded node

http://www.cs.duke.edu/courses/fall08/cps270/
Properties of breadth-first search

- **Completeness**
  - Complete (if \( b \) is finite)

- **Time complexity**
  - \( 1 + b + b^2 + b^3 + ... + b^d = b^{d+1} - 1 / (b - 1) = O(b^d) \), i.e., exponential in \( d \)
    - \( b \) = branching factor, \( d \) = depth
    - equals the number of nodes visited during BFS

- **Space complexity**
  - \( O(b^d) \) (keeps every node in memory)
  - Space is the big problem; can easily generate nodes at 1MB/sec, so 24hrs = 86GB

- **Optimality**
  - Optimal if cost = 1 per step (not optimal in general)
Uniform-cost search

Expand least-cost unexpanded node
Properties of uniform-cost search

• **Completeness**  
  – Complete if step cost $\geq \epsilon$

• **Time complexity**  
  – # of nodes with $g \leq$ cost of optimal solution

• **Space complexity**  
  – # of nodes with $\leq$ cost of optimal solution

• **Optimality**  
  – Optimal
Depth-first search

Expand deepest unexpanded node

http://www.cs.duke.edu/courses/fall08/cps270/
Properties of depth-first search

• **Completeness**
  – Not complete – fails in infinite-depth space, spaces with loops
  – Modify to avoid repeated states along path
    • complete in finite spaces

• **Time complexity**
  – $O(b^m)$ – terrible if $m$ is much larger than $d$
  – but if solutions are dense, may be much faster than bread-first

• **Space complexity**
  – $O(bm)$ – linear space

• **Optimality**
  – Not optimal
Iterative Deepening

• Advantage
  – Linear memory requirements of depth-first search
  – Guarantee for goal node of minimal depth

• Procedure
  – Successive depth-first searches are conducted – each with depth bounds increasing by 1
Iterative Deepening (Cont’d)

Figure 8.5 Stages in Iterative-Deepening Search
Iterative Deepening (Cont’d)

• The number of nodes
  – In case of breadth-first search
    \[ N_{\text{bf}} = 1 + b + b^2 + \cdots + b^d = \frac{b^{d+1} - 1}{b - 1} \] (b: branching factor, d: depth)
  
  – In case of iterative deepening search
    \[ N_{\text{id}} = \sum_{j=0}^{d} \frac{b^{j+1} - 1}{b - 1} \]
    \[ = \frac{1}{b - 1} \left[ b \left( \sum_{j=0}^{d} b^j \right) - \sum_{j=0}^{d} 1 \right] = \frac{1}{b - 1} \left[ b \left( \frac{b^{d+1} - 1}{b - 1} \right) - (d + 1) \right] \]
    \[ = \frac{b^{d+2} - 2b - bd + d + 1}{(b - 1)^2} \]
Iterative Deepening (Cont’d)

– For large $d$ the ratio $N_{id}/N_{df}$ is $b/(b-1)$
– For a branching factor of 10 and deep goals, 11% more nodes expansion in iterative-deepening search than breadth-first search
– Related technique *iterative broadening* is useful when there are many goal nodes
Informed Search

• So far, have assumed that no nongoal state looks better than another

• Unrealistic
  – Even without knowing the road structure, some locations seem closer to the goal than others
  – Some states of the 8s puzzle seem closer to the goal than others

• Makes sense to expand closer-seeming nodes first
Heuristics

• Key notion: **heuristic function** $h(n)$ gives an estimate of the distance from $n$ to the goal
  – $h(n)=0$ for goal nodes

• E.g. **straight-line distance** for traveling problem

  ![Graph](http://www.cs.duke.edu/courses/fall08/cps270/)

  - $h(A) = 9$, $h(B) = 8$, $h(C) = 9$, $h(D) = 6$, $h(E) = 3$, $h(F) = 0$
Greedy best-first search

- Expand nodes with lowest h values first
- Rapidly finds the optimal solution

State A:
- Cost: 0
- Heuristic: 9

State B:
- Cost: 3
- Heuristic: 8

State C:
- Cost: 3
- Heuristic: 6

State D:
- Cost: 3
- Heuristic: 6

State E:
- Cost: 7
- Heuristic: 3

State F:
- Cost: 11
- Heuristic: 0

Goal state!
A bad example for greedy

h(A) = 9, h(B) = 5, h(D) = 6, h(E) = 3, h(F) = 0

Problem: greedy evaluates the promise of a node only by how far is left to go, does not take cost occurred already into account
Properties of greedy search

- **Completeness**
  - Not complete – can get stuck in loops
  - Complete in finite space with repeated-state checking

- **Time complexity**
  - $O(b^m)$ – but a good heuristic can give dramatic improvement

- **Space complexity**
  - $O(b^m)$ – keeps all nodes in memory

- **Optimality**
  - Not optimal
Algorithm A*

- Algorithm A*
  - Reorders the nodes on OPEN according to increasing values of $\hat{f}$

- Some additional notation
  - $h(n)$: the *actual cost* of the minimal cost path between $n$ and a goal node
  - $g(n)$: the cost of a minimal cost path from $n_0$ to $n$
  - $f(n) = g(n) + h(n)$: the cost of a minimal cost path from $n_0$ to a goal node over all paths via node $n$
  - $f(n_0) = h(n_0)$: the cost of a minimal cost path from $n_0$ to a goal node
  - $\hat{h}(n)$: estimate of $h(n)$
  - $\hat{g}(n)$: the cost of the lowest-cost path found by A* so far to $n$
Algorithm A* (Cont’d)

• Algorithm A*
  – If $\hat{h} = 0$: uniform-cost search
  – When the graph being searched is not a tree?
    • more than one sequence of actions that can lead to the same world state from the starting state
  – In 8-puzzle problem
    • Actions are reversible: implicit graph is not a tree
    • Ignore loops in creating 8-puzzle search tree: don’t include the parent of a node among its successors
    • Step 6
      – Expand $n$, generating a set $M$ of successors that are not already parents (ancestors) of $n$ + install $M$ as successors of $n$ by creating arcs from $n$ to each member of $M$
A* search

- Let $g(n)$ be cost incurred already on path to $n$
- Expand nodes with lowest $g(n) + h(n)$ first

- $h(A) = 9$, $h(B) = 5$, $h(D) = 6$, $h(E) = 3$, $h(F) = 0$
- Note: if $h=0$ everywhere, then just uniform cost search
Figure 9.3 Heuristic Search Notation

Start node, $n_0$

$f(n_0) =$ cost of lowest-cost (optimal) path to a goal

$g(n) =$ cost of best path from $n_0$ to $n$

$h(n) =$ cost of optimal path from $n$ to a goal

$\hat{f}, \hat{g}, \hat{h}$ are estimates of $f, g, h$, respectively

$$\hat{f} = \hat{g} + \hat{h}$$
Properties of A*

• **Completeness**
  – Complete, unless there are infinitely many nodes with $f \leq f(G)$

• **Time complexity**
  – Exponential in [relative error in $h$ x length of soln.]

• **Space complexity**
  – Keeps all nodes in memory

• **Optimality**
  – Optimal – cannot expand $f_{i+1}$ until $f_i$ is finished.
Algorithm A* (Cont’d)

• A* that maintains the search graph
1. Create a search graph, $G$, consisting solely of the start node, $n_0 \rightarrow$ put $n_0$ on a list $OPEN$
2. Create a list $CLOSED$: initially empty
3. If $OPEN$ is empty, exit with failure
4. Select the first node on $OPEN \rightarrow$ remove it from $OPEN \rightarrow$ put it on $CLOSED$: node $n$
5. If $n$ is a goal node, exit successfully: obtain solution by tracing a path along the pointers from $n$ to $n_0$ in $G$
6. Expand node $n$, generating the set, $M$, of its successors that are not already ancestors of $n$ in $G \rightarrow$ install these members of $M$ as successors of $n$ in $G$
Algorithm A* (Cont’d)

7. Establish a pointer to $n$ from each of those members of $M$ that were not already in $G \rightarrow$ add these members of $M$ to OPEN
   \rightarrow for each member, $m$, redirect its pointer to $n$ if the best path to $m$ found so far is through $n \rightarrow$ for each member of $M$ already on CLOSED, redirect the pointers of each of its descendants in $G$

8. Reorder the list OPEN in order of increasing $f$ values

9. Go to step 3

– Redirecting pointers of descendants of nodes
  • Save subsequent search effort
Admissibility

• A heuristic is **admissible** if it never overestimates the distance to the goal
  – If n is the optimal solution reachable from n’, then \( g(n) \geq g(n') + h(n') \)
• Straight-line distance is admissible: can’t hope for anything better than a straight road to the goal
• Admissible heuristic means that A* is always optimistic
Optimality of A*

• If the heuristic is admissible, A* is optimal (in the sense that it will never return a suboptimal solution)

• Proof:
  – Suppose a suboptimal solution node n with solution value C > C* is about to be expanded (where C* is optimal)
  – Let n* be an optimal solution node (perhaps not yet discovered)
  – There must be some node n’ that is currently in the fringe and on the path to n*
  – We have g(n) = C > C* = g(n*) ≥ g(n’) + h(n’)
  – But then, n’ should be expanded first (contradiction)
Figure 9.6 Relationships Among Search Algorithm
A* is not complete (in contrived examples)

- No optimal search algorithm can succeed on this example (have to keep looking down the path in hope of suddenly finding a solution)

infinitely many nodes on a straight path to the goal that doesn’t actually reach the goal
A* is optimally efficient

- A* is **optimally efficient** in the sense that any other optimal algorithm must expand at least the nodes A* expands.

Proof:
- Besides solution, A* expands exactly the nodes with \( g(n) + h(n) < C^* \)
  
  - Assuming it does not expand non-solution nodes with \( g(n) + h(n) = C^* \)

  - Any other optimal algorithm must expand at least these nodes (since there may be a better solution there)

Note: This argument assumes that the other algorithm uses the same heuristic \( h \)
**Heuristic Function**

- Function $h(N)$ that estimates the cost of the cheapest path from node $N$ to goal node.
- Example: 8-puzzle

![8-puzzle example](http://www.cs.umd.edu/class/spring2013/cmsc421/)

$h(N) =$ number of misplaced tiles
$= 6$

or,

$h(N) =$ sum of the distances of every tile to its goal position
$= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1$
$= 13$
8-puzzle

\[ f(N) = h(N) = \text{number of misplaced tiles} \]
8-puzzle

\[ f(N) = \text{depth} + \text{h}(N) \]

with \( \text{h}(N) = \) number of misplaced tiles
8-puzzle

\[ f(N) = h(N) = \sum \text{distances of tiles to goal} \]
Consistent Heuristic

• The admissible heuristic $h$ is **consistent** (or satisfies the **monotone restriction**) if for every node $N$ and every successor $N'$ of $N$:

$$h(N) \leq c(N,N') + h(N')$$

(triangular inequality)

• In 8-puzzle problem,
  
  $h_1(N) =$ number of misplaced tiles
  
  $h_2(N) =$ sum of distances of each tile to goal

are both consistent
Avoiding Repeated States in A*

If the heuristic $h$ is consistent, then:

• Let CLOSED be the list of states associated with expanded nodes

• When a new node $N$ is generated:
  – If its state is in CLOSED, then discard $N$
  – If it has the same state as another node in the fringe, then discard the node with the largest $f$
Heuristic Accuracy

• $h(N) = 0$ for all nodes is admissible and consistent. Hence, breadth-first and uniform-cost are particular $A^*$ !!!

• Let $h_1$ and $h_2$ be two admissible and consistent heuristics such that for all nodes $N$: $h_1(N) \leq h_2(N)$.

• Then, every node expanded by $A^*$ using $h_2$ is also expanded by $A^*$ using $h_1$.

• $h_2$ is more informed than $h_1$
Iterative-Deepening A*

• Breadth-first search
  – Exponentially growing memory requirements

• Iterative deepening search
  – Memory grows linearly with the depth of the goal
  – Parallel implementation of IDA*: further efficiencies gain

• IDA*
  – Cost cut off in the first search: $\hat{f}(n_0) = \hat{g}(n_0) + \hat{h}(n_0) = \hat{h}(n_0)$
  – Depth-first search with backtracking
  – If the search terminates at a goal node: minimal-cost path
  – Otherwise
    • increase the cut-off value and start another search
    • The lowest $\hat{f}$ values of the nodes visited (not expanded) in the previous search is used as the new cut-off value in the next search
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) = \) number of misplaced tiles

Cutoff=4

http://www.cs.umd.edu/class/spring2013/cmsc421/
8-Puzzle

\[ f(N) = g(N) + h(N) \]
with \[ h(N) \] = number of misplaced tiles

Cutoff=4

\[ \begin{array}{c}
\begin{array}{c}
\text{4} \\
\text{6}
\end{array}
\end{array} \]

\[ \begin{array}{c}
\begin{array}{c}
\text{4} \\
\text{6}
\end{array}
\end{array} \]

\[ \begin{array}{c}
\begin{array}{c}
\text{4} \\
\text{6}
\end{array}
\end{array} \]
8-Puzzle

\[ f(N) = g(N) + h(N) \]
with \( h(N) = \) number of misplaced tiles
8-Puzzle

\[ f(N) = g(N) + h(N) \]
with \( h(N) \) = number of misplaced tiles

Cutoff=4
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) \) = number of misplaced tiles

Cutoff=4

http://www.cs.umd.edu/class/spring2013/cmsc421/
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) \) = number of misplaced tiles

Cutoff=5
8-Puzzle

\[ f(N) = g(N) + h(N) \]
with \( h(N) \) = number of misplaced tiles
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) \) = number of misplaced tiles

Cutoff=5

http://www.cs.umd.edu/class/spring2013/cmsc421/
8-Puzzle

\[ f(N) = g(N) + h(N) \]
with \( h(N) = \text{number of misplaced tiles} \)
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) = \) number of misplaced tiles
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) \) = number of misplaced tiles
8-Puzzle

\[ f(N) = g(N) + h(N) \]

with \( h(N) = \) number of misplaced tiles
About Heuristics

• Heuristics are intended to orient the search along promising paths
• The time spent computing heuristics must be recovered by a better search
• After all, a heuristic function could consist of solving the problem; then it would perfectly guide the search
• Deciding which node to expand is sometimes called meta-reasoning
• Heuristics may not always look like numbers and may involve large amount of knowledge
What's the Issue?

• Search is an iterative **local** procedure

• Good heuristics should provide some **global look-ahead** (at low computational cost)