BIL 682 – Artificial Intelligence

Week #2: Solving problems by searching

Asst. Prof. Aykut Erdem
Dept. of Computer Engineering
Hacettepe University
Today

• Search problems
• Uninformed search
• Informed (heuristic) search

The slides are mostly adopted from Dan Klein (UC Berkeley), Lana Lazebnik (UNC) and Hal Daumé III (UMD).
Announcement

• The first problem set is up!
  – Due Tuesday, 2013-03-26, 11:59 PM.

• Solving mazes by search
  The purpose of the problem set is to familiarize you with solving problems by uninformed and informed search approaches.

Late policy

You may use up to five extension days (in total) over the course of the semester for the three PSets. Any additional unapproved late submission will be weighted by 0.5.
Reflex Agents

- Reflex Agents:
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world’s current state
  - Do not consider the future consequences of their actions
    - Act on how the world IS
- Can a reflex agent be rational?
Goal Based Agents

• Goal-based agents:
  – Plan ahead
  – Decisions based on (hypothesized) consequences of actions
  – Must have a model of how the world evolves in response to actions
  – Act on how the world WOULD BE
A search problem consists of:

- A state space
- A successor function
- A start state and a goal test

A solution is a sequence of actions (a plan) which transforms the start state to a goal state.

The performance measure is defined by (a) reaching the goal and (b) how “expensive” the path to the goal is.
Search problem components

• Initial state
• Actions
• Transition model
  – What is the result of performing a given action in a given state?
• Goal state
• Path cost
  – Assume that it is a sum of nonnegative step costs
• The **optimal solution** is the sequence of actions that gives the lowest path cost for reaching the goal
Example: Romania (Route Finding Problem)

- On vacation in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

- **Initial state**
  - Arad

- **Actions**
  - Go from one city to another

- **Transition model**
  - If you go from city A to city B, you end up in city B

- **Goal state**
  - Bucharest

- **Path cost**
  - Sum of edge costs
State space

• The initial state, actions, and transition model define the **state space** of the problem
  – The set of all states reachable from initial state by any sequence of actions
  – Can be represented as a **directed graph** where the nodes are states and links between nodes are actions

• What is the state space for the Romania problem?
Example: Vacuum world

- **States**
  - Agent location and dirt location
  - How many possible states?
  - What if there are $n$ possible locations?

- **Actions**
  - Left, right, suck

- **Transition model**
Example: The 8-puzzle

• **States**
  – Locations of tiles
    • 8-puzzle: 181,440 states
    • 15-puzzle: 1.3 trillion states
    • 24-puzzle: $10^{25}$ states

• **Actions**
  – Move blank left, right, up, down

• **Path cost**
  – 1 per move

• **Finding the optimal solution of n-Puzzle is NP-hard**
Example: Robot motion planning

- **States**
  - Real-valued coordinates of robot joint angles

- **Actions**
  - Continuous motions of robot joints

- **Goal state**
  - Desired final configuration (e.g., object is grasped)

- **Path cost**
  - Time to execute, smoothness of path, etc.
State Space Graphs

• **State space graph:**
  A mathematical representation of a search problem
  – For every search problem, there’s a corresponding state space graph
  – The successor function is represented by arcs

• We can rarely build this graph in memory (so we don’t)
Example: Vacuum World State Space Graph
Example: Romania State Space Graph
Another Search Tree

• Search:
  – Expand out possible plans
  – Maintain a fringe of unexpanded plans
    • Fringe (Frontier): the collection of nodes that have been generated but not yet been expanded
  – Try to expand as few tree nodes as possible
States vs. Nodes

- **Problem graphs have problem states**
  - Represent an abstracted state of the world
  - Have successors, predecessors, can be goal / non-goal

- **Search trees have search nodes**
  - Represent a plan (path) which results in the node’s state
  - Have 1 parent, a length and cost, point to a problem state
  - Expand uses successor function to create new tree nodes
  - The same problem state in multiple search tree nodes
Search tree

- “What if” tree of possible actions and outcomes
- The root node corresponds to the starting state
- The children of a node correspond to the successor states of that node’s state
- A path through the tree corresponds to a sequence of actions
  - A solution is a path ending in the goal state
- Nodes vs. states
  - A state is a representation of a physical configuration, while a node is a data structure that is part of the search tree
Tree Search Algorithm Outline

function Tree-Search(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree

• Initialize the frontier using the starting state
• While the frontier is not empty
  – Choose a frontier node to expand according to search strategy
  – If the node contains the goal state, return solution
  – Else expand the node and add its children to the frontier
Tree Search Example
Tree Search Example
Tree Search Example

Frontier
Search strategies

- A **search 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?
  - **Optimality**: does it always find a least-cost solution?
  - **Time complexity**: number of nodes generated
  - **Space complexity**: maximum number of nodes in memory

- Time and space complexity are measured in terms of:
  - $b$: maximum branching factor of the search tree
  - $d$: depth of the optimal solution
  - $m$: maximum length of any path in the state space (may be infinite)
Review: Search Problem Formulation

- Initial state
- Actions
- Transition model
- Goal state (or goal test)
- Path cost

- What is the optimal solution?
- What is the state space?
Today

- Search problems
- Uninformed search
- Informed (heuristic) search
Uninformed search strategies

• Uninformed search strategies use only the information available in the problem definition and do not guide the search with any additional information about the problem.
  – Breadth-first search
  – Uniform-cost search
  – Depth-first search
  – Iterative deepening search
Breadth-first search

• Expand shallowest unexpanded node

• Implementation:
  – *frontier* is a FIFO queue, i.e., new successors go at end

\[ \text{D} \quad \text{B} \quad \text{A} \quad \text{C} \quad \text{G} \quad \text{E} \quad \text{F} \]
Breadth-first search

• Expand shallowest unexpanded node

• Implementation:
  – *frontier* is a FIFO queue, i.e., new successors go at end
Breadth-first search

• Expand shallowest unexpanded node

• Implementation:
  – *frontier* is a FIFO queue, i.e., new successors go at end
Breadth-first search

• Expand shallowest unexpanded node

• Implementation:
  – *frontier* is a FIFO queue, i.e., new successors go at end
Breadth-first search

• Expand shallowest unexpanded node

• Implementation:
  – *frontier* is a FIFO queue, i.e., new successors go at end
Properties of breadth-first search

• **Complete?**
  Yes (if branching factor $b$ is finite)

• **Optimal?**
  Yes – if cost = 1 per step

• **Time?**
  Number of nodes in a $b$-ary tree of depth $d$: $O(b^d)$
  ($d$ is the depth of the optimal solution)
  $$1+b+b^2+b^3+\ldots+b^d + b(b^d-1) = O(b^{d+1})= O(b^d)$$

• **Space?**
  $O(b^d)$

• Space is the bigger problem (more than time)
Uniform-cost search

• For each frontier node, save the total cost of the path from the initial state to that node
• Expand the frontier node with the lowest path cost
• Implementation: *frontier* is a priority queue ordered by path cost
• Equivalent to breadth-first if step costs all equal
Illustration of uniform cost search
(same as Dijkstra’s shortest path algorithm)

Properties of uniform-cost search

• **Complete?**
  Yes, if step cost is greater than some positive constant $\varepsilon$ (we don’t want infinite sequences of steps that have a finite total cost)

• **Optimal?**
  Yes – nodes expanded in increasing order of path cost

• **Time?**
  Number of nodes with path cost $\leq$ cost of optimal solution ($C^*$), $O(b^{C^*/\varepsilon})$
  This can be greater than $O(b^d)$: the search can explore long paths consisting of small steps before exploring shorter paths consisting of larger steps

• **Space?**
  $O(b^{C^*/\varepsilon})$
Depth-first search

• Expand deepest unexpanded node
• Implementation:
  – *frontier* = LIFO queue, i.e., put successors at front
Depth-first search

• Expand deepest unexpanded node
• Implementation:
  – *frontier* = LIFO queue, i.e., put successors at front
Depth-first search

• Expand deepest unexpanded node
• Implementation:
  – *frontier* = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - \textit{frontier} = LIFO queue, i.e., put successors at front
Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front
Depth-first search

• Expand deepest unexpanded node
• Implementation:
  – *frontier* = LIFO queue, i.e., put successors at front
Depth-first search

• Expand deepest unexpanded node
• Implementation:
  – \textit{frontier} = LIFO queue, i.e., put successors at front
Depth-first search

• Expand deepest unexpanded node
• Implementation:
  – *frontier* = LIFO queue, i.e., put successors at front
Properties of depth-first search

• **Complete?**
  Fails in infinite-depth spaces, spaces with loops
  Modify to avoid repeated states along path
  \( \rightarrow \) complete in finite spaces

• **Optimal?**
  No – returns the first solution it finds

• **Time?**
  Could be the time to reach a solution at maximum depth \( m: O(b^m) \)
  Terrible if \( m \) is much larger than \( d \)
  But if there are lots of solutions, may be much faster than BFS

• **Space?**
  \( O(bm) \), i.e., linear space!
Iterative deepening search

• Use DFS as a subroutine
  
  1. Check the root
  2. Do a DFS searching for a path of length 1
  3. If there is no path of length 1, do a DFS searching for a path of length 2
  4. If there is no path of length 2, do a DFS searching for a path of length 3…
Iterative deepening search

Limit = 0
Iterative deepening search
Iterative deepening search

Limit = 2
Iterative deepening search

Limit = 3
Properties of iterative deepening search

• **Complete?**
  Yes

• **Optimal?**
  Yes, if step cost = 1

• **Time?**
  \[(d+1)b^0 + d b^1 + (d-1)b^2 + ... + b^d = O(b^d)\]

• **Space?**
  \[O(bd)\]
Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one!
Handling repeated states

- Initialize the frontier using the starting state
- While the frontier is not empty
  - Choose a frontier node to expand according to search strategy
  - If the node contains the goal state, return solution
  - Else expand the node and add its children to the frontier

- To handle repeated states:
  - Keep an explored set; add each node to the explored set every time you expand it
  - Every time you add a node to the frontier, check whether it already exists in the frontier with a higher path cost, and if yes, replace that node with the new one
## Review: Uninformed search strategies

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete?</th>
<th>Optimal?</th>
<th>Time complexity</th>
<th>Space complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Yes</td>
<td>If all step costs are equal</td>
<td>$O(b^d)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>UCS</td>
<td>Yes</td>
<td>Yes</td>
<td>Number of nodes with $g(n) \leq C^*$</td>
<td></td>
</tr>
<tr>
<td>DFS</td>
<td>No</td>
<td>No</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>IDS</td>
<td>Yes</td>
<td>If all step costs are equal</td>
<td>$O(b^d)$</td>
<td>$O(bd)$</td>
</tr>
</tbody>
</table>

- **b:** maximum branching factor of the search tree
- **d:** depth of the optimal solution
- **m:** maximum length of any path in the state space
- **C***: cost of optimal solution
- **g(n):** cost of path from start state to node n
Today

• Search problems
• Uninformed search
• Informed (heuristic) search
Informed search

• Idea: give the algorithm “hints” about the desirability of different states
  – Use an evaluation function to rank nodes and select the most promising one for expansion

• Greedy best-first search
• A* search
Heuristic function

• Heuristic function $h(n)$ estimates the cost of reaching goal from node $n$

• Example:
Heuristic for the Romania problem
Greedy best-first search

• Expand the node that has the lowest value of the heuristic function $h(n)$
Greedy best-first search example
Greedy best-first search example
Greedy best-first search example
Greedy best-first search example
Properties of greedy best-first search

• Complete?
  No – can get stuck in loops
Properties of greedy best-first search

- **Complete?**
  No – can get stuck in loops

- **Optimal?**
  No
Properties of greedy best-first search

- **Complete?**
  No – can get stuck in loops

- **Optimal?**
  No

- **Time?**
  Worst case: $O(b^m)$
  Can be much better with a good heuristic

- **Space?**
  Worst case: $O(b^m)$
How can we fix the greedy problem?
A* search

• Idea: avoid expanding paths that are already expensive
• The **evaluation function** \( f(n) \) is the estimated total cost of the path through node \( n \) to the goal:

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

- **\( g(n) \):** cost so far to reach \( n \) (path cost)
- **\( h(n) \):** estimated cost from \( n \) to goal (heuristic)
- **\( f(n) \):** estimated total cost of path through \( n \) to goal
A* search example
A* search example
A* search example
A* search example
A* search example
A* search example
Another example

Admissible heuristics

• A heuristic $h(n)$ is admissible if for every node $n$, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from $n$

• An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic

• Example: straight line distance never overestimates the actual road distance

• **Theorem:** If $h(n)$ is admissible, A* using tree search is optimal
Optimality of A*

• Suppose A* search terminates at goal state $n^*$ with $f(n^*) = g(n^*) = C^*$
• For any other frontier node $n$, we have $f(n) \geq C^*$
• In other words, the estimated cost $f(n)$ of any solution path going through $n$ is no lower than $C^*$
• Since $f(n)$ is an optimistic estimate, there is no way that a solution path going through $n$ can have an actual cost lower than $C^*$
Optimality of A*

- A* is *optimally efficient* – no other tree-based algorithm that uses the same heuristic can expand fewer nodes and still be guaranteed to find the optimal solution
  - Any algorithm that does not expand all nodes with \( f(n) \leq C^* \) risks missing the optimal solution
Properties of A* 

- **Complete?**
  Yes – unless there are infinitely many nodes with $f(n) \leq C^*$

- **Optimal?**
  Yes

- **Time?**
  Number of nodes for which $f(n) \leq C^*$ (exponential)

- **Space?**
  Exponential
Designing heuristic functions

• Heuristics for the 8-puzzle

\[ h_1(n) = \text{number of misplaced tiles} \]

\[ h_2(n) = \text{total Manhattan distance (number of squares from desired location of each tile)} \]

\[ h_1(\text{start}) = 8 \]

\[ h_2(\text{start}) = 3+1+2+2+2+3+3+2 = 18 \]

• Are \( h_1 \) and \( h_2 \) admissible?
Heuristics from 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 $h_1(n)$ gives the shortest solution
• If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution
Heuristics from subproblems

- Let $h_3(n)$ be the cost of getting a subset of tiles (say, 1,2,3,4) into their correct positions.
- Can precompute and save the exact solution cost for every possible subproblem instance – *pattern database*.
Dominance

• If $h_1$ and $h_2$ are both admissible heuristics and $h_2(n) \geq h_1(n)$ for all $n$, (both admissible) then $h_2$ dominates $h_1$

• Which one is better for search?
  – A* search expands every node with $f(n) < C^*$ or $h(n) < C^* - g(n)$
  – Therefore, A* search with $h_1$ will expand more nodes
Dominance

• Typical search costs for the 8-puzzle (average number of nodes expanded for different solution depths):

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

  • $d=24$  
    IDS  $\approx 54,000,000,000$ nodes  
    $A^*(h_1) = 39,135$ nodes  
    $A^*(h_2) = 1,641$ nodes
Combining heuristics

• Suppose we have a collection of admissible heuristics $h_1(n), h_2(n), \ldots, h_m(n)$, but none of them dominates the others

• How can we combine them?

\[ h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\} \]
Weighted A* search

• **Idea:** speed up search at the expense of optimality

• Take an admissible heuristic, “inflate” it by a multiple \( \alpha > 1 \), and then perform A* search as usual

• Fewer nodes tend to get expanded, but the resulting solution may be suboptimal (its cost will be at most \( \alpha \) times the cost of the optimal solution)
Example of weighted A* search

Heuristic: 5 * Euclidean distance from goal
Example of weighted A* search

Heuristic: 5 * Euclidean distance from goal

Compare: Exact A*
Memory-bounded search

• The memory usage of A* can still be exorbitant
• How to make A* more memory-efficient while maintaining completeness and optimality?

• Iterative deepening A* search
• Recursive best-first search, SMA*
  – Forget some subtrees but remember the best f-value in these subtrees and regenerate them later if necessary

• Problems: memory-bounded strategies can be complicated to implement, suffer from “thrashing”
## All search strategies

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete?</th>
<th>Optimal?</th>
<th>Time complexity</th>
<th>Space complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Yes</td>
<td>If all step costs are equal</td>
<td>O(b^d)</td>
<td>O(b^d)</td>
</tr>
<tr>
<td>UCS</td>
<td>Yes</td>
<td>Yes</td>
<td>Number of nodes with g(n) ≤ C*</td>
<td></td>
</tr>
<tr>
<td>DFS</td>
<td>No</td>
<td>No</td>
<td>O(b^m)</td>
<td>O(bm)</td>
</tr>
<tr>
<td>IDS</td>
<td>Yes</td>
<td>If all step costs are equal</td>
<td>O(b^d)</td>
<td>O(bd)</td>
</tr>
</tbody>
</table>
| Greedy    | No        | No       | Worst case: O(b^m)  
Best case: O(bd) | |
| A*        | Yes       | Yes      | Number of nodes with g(n)+h(n) ≤ C* | |
Two types of search problems

Rubik’s Cube

• Start state is given
• Goal state is known ahead of time
• Solution path matters

N-Queens Problem

• No specific start state
• Goal state is unknown (only have goal test)
• Solution path does not matter
Local search algorithms

• Some types of search problems can be formulated in terms of optimization
  – We don’t have a start state, don’t care about the path to a solution, can move around state space arbitrarily
  – We have an objective function that tells us about the quality of a state (possible solution), and we want to find a good solution by minimizing or maximizing the value of this function
Example: $n$-queens problem

• Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

• **State space:** all possible $n$-queen configurations

• What’s the **objective function**?
  – Number of pairwise conflicts
Example: Traveling salesman problem

- Find the shortest tour connecting a given set of sites
- **State space:** all possible tours
- **Objective function:** length of tour
Hill-climbing (greedy) search

• Idea: keep a single “current” state and try to locally improve it
• “Like climbing mount Everest in thick fog with amnesia”
Example: $n$-queens problem

- Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
- **State space:** all possible $n$-queen configurations
- **Objective function:** number of pairwise conflicts
- What’s a possible local improvement strategy?
  - Move one queen within its column to reduce conflicts
Example: Traveling Salesman Problem

- Find the shortest tour connecting n cities
- **State space:** all possible tours
- **Objective function:** length of tour
- What’s a possible local improvement strategy?
  - Start with any complete tour, exchange endpoints of two edges
Hill-climbing (greedy) search (maximization)

• Initialize current to starting state
• Loop:
  – Let next = highest-valued successor of current
  – If value(next) < value(current) return current
  – Else let current = next

• Variants: choose first better successor, randomly choose among better successors
Hill-climbing search

• Is it complete/optimal?
  – No – can get stuck in local optima
  – Example: local optimum for the 8-queens problem

\[ h = 1 \]
The state space “landscape”

- How to escape local maxima?
  - Random restart hill-climbing
Local beam search

• Start with $k$ randomly generated states
• At each iteration, all the successors of all $k$ states are generated
• If any one is a goal state, stop; else select the $k$ best successors from the complete list and repeat

• Not the same as running $k$ greedy searches in parallel!
Simulated annealing search

• Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency
  – Probability of taking downhill move decreases with number of iterations, steepness of downhill move
  – Controlled by annealing schedule

• Inspired by tempering of glass, metal
Simulated annealing search

- Initialize $current$ to starting state
- For $i = 1$ to $\infty$
  - If $T(i) = 0$ return $current$
  - Let $next = \text{random successor of } current$
  - Let $\Delta = \text{value}(next) - \text{value}(current)$
  - If $\Delta > 0$ then let $current = next$
  - Else let $current = next$ with probability $\exp(\Delta/T(i))$
Effect of temperature

\[ \exp\left(\frac{\Delta}{T}\right) \]
Simulated annealing search

• One can prove: If temperature decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching one

• However:
  – This usually takes impractically long
  – The more downhill steps you need to escape a local optimum, the less likely you are to make all of them in a row

• More modern techniques: general family of *Markov Chain Monte Carlo* (MCMC) algorithms for exploring complicated state spaces
Gene\+algorithms use a natural selection metaphor

Like beam search (selection), but also have pairwise crossover operators, with optional mutation

Probably the most misunderstood, misapplied (and even maligned) technique around!
Example: N-Queens

- Why does crossover make sense here?
- When wouldn’t it make sense?
- What would mutation be?
- What would a good fitness function be?
Continuous Problems

- Placing airports in Romania
  - States \((x_1,y_1,x_2,y_2,x_3,y_3)\)
  - Cost: sum of squared distances to closest city
Gradient Methods

• How to deal with continuous (therefore infinite) state spaces?
• Discretization: bucket ranges of values
  – e.g. force integral coordinates
• Continuous optimization
  – e.g. gradient ascent

$$\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

$$x \leftarrow x + \alpha \nabla f(x)$$