CSC242: Artificial Intelligence
Lecture 4
Local Search
Upper Level Writing

Topics due to me by next class!

First draft due Mar 4

Goal: final paper 15 pages +/- 2 pages

12 pt font, 1.5 line spacing

Get in touch early!
Assignments

- Homework 1
  - Will be posted on BB tonight
  - Solutions will be given out (hardcopy) in class Tuesday

- Project 1: Othello
  - Will be posted Tuesday night
Local Search
States + Actions + Transition Model

=  
State Space

The set of all states reachable from the initial state by some sequence of actions
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
## Search Strategies

<table>
<thead>
<tr>
<th>Complete?</th>
<th>BFS</th>
<th>DFS</th>
<th>IDS</th>
<th>Greedy</th>
<th>A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Optimal?</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^d)$</td>
<td>$O(b^m)$</td>
<td>$O(b^d)$</td>
<td>$O(b^m)$</td>
<td>$O(b^{cd})$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^d)$</td>
<td>$O(bm)$</td>
<td>$O(bd)$</td>
<td>$O(b^m)$</td>
<td>$O(b^d)$</td>
</tr>
</tbody>
</table>

* If step costs are identical
† With an admissible heuristic
Systematic Search

• Enumerates paths from initial state
• Records what alternatives have been explored at each point in the path
Systematic Search

- Enumerates paths from initial state
- Records what alternatives have been explored at each point in the path

Good:
Systematic Search

- Enumerates paths from initial state
- Records what alternatives have been explored at each point in the path

Good: Systematic $\rightarrow$ Exhaustive
Systematic Search

- Enumerates paths from initial state
- Records what alternatives have been explored at each point in the path

Good: Systematic $\rightarrow$ Exhaustive

Bad:
Systematic Search

- Enumerates paths from initial state
- Records what alternatives have been explored at each point in the path

Good: Systematic $\rightarrow$ Exhaustive

Bad: Exponential time and/or space
The Problem Last Class!
Why are Java applications blocked by your security settings with the latest Java?

This article applies to:
- Java version(s): 7.0

SYMPTOMS

Trying to run the Java applications with Java version 7 Update 51, generates messages that says

- Java applications are blocked by your security settings.
- Missing Application-Name manifest attribute
- Missing required Permissions manifest attribute in main jar

CAUSE

Starting with Java 7 Update 51, Java has enhanced security model to make user system less vulnerable to the external exploits. The new version of Java does not allow users to run the applications that are not signed (Unsigned), Self signed (not signed by trusted authority) and the applications that are missing permission attributes.
N-Queens as State-Space Search Problem
N-Queens as State-Space Search Problem

- State
N-Queens as State-Space Search Problem

- State
- Actions
N-Queens as State-Space Search Problem

- State
- Actions
- Transition Model
N-Queens as State-Space Search Problem

• State
• Actions
• Transition Model
• Initial State
N-Queens as State-Space Search Problem

- State
- Actions
- Transition Model
- Initial State
- Goal State(s)/Test
N-Queens as State-Space Search Problem

- State
- Actions
- Transition Model
- Initial State
- Goal State(s)/Test
- Step costs
Local Search
Local Search

- Evaluates and modifies a small number of current states
- Does not record history of search (paths, explored set, etc.)
Local Search

• Evaluates and modifies a small number of current states
• Does not record history of search (paths, explored set, etc.)

Good:
Local Search

- Evaluates and modifies a small number of current states
- Does not record history of search (paths, explored set, etc.)

Good: Very little (constant) memory
Local Search

• Evaluates and modifies a small number of current states
• Does not record history of search (paths, explored set, etc.)

Good: Very little (constant) memory

Bad:
Local Search

• Evaluates and modifies a small number of current states

• Does not record history of search (paths, explored set, etc.)

Good: Very little (constant) memory

Bad: May not explore all alternatives
Local Search

• Evaluates and modifies a small number of current states

• Does not record history of search (paths, explored set, etc.)

Good: Very little (constant) memory

Bad: May not explore all alternatives

=> Incomplete
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}

State localSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
Solution graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}

State localSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {

        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
Solution: graphSearch(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());
    Set<Node> explored = new Set<Node>();
    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return node.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}

State localSearch(Problem p) {
    while (true) {
        if (p.isGoalState(node.getState())) {
        }
        for (Node n : node.expand()) {
        }
    }
}
State localSearch(Problem p) {
    Node node = new Node(p.getInitialState());
    while (true) {
        if (p.isGoalState(node.getState())) {
            return;
        }
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
State localSearch(Problem p) {
    Node node = new Node(p.getInitialState());
    while (true) {
        if (p.isGoalState(node.getState())) {
            return node.getState();
        }
        for (Node n : node.expand()) {
            return n.getState();
        }
    }
}
State localSearch(Problem p) {
    Node node = new Node(p.getInitialState());
    while (true) {
        if (p.isGoalState(node.getState())) {
            return node.getState();
        }
        for (Node n : node.expand()) {
            ???
        }
    }
}
$h(n)$
State localSearch(Problem p) {
    Node node = new Node(p.getInitialState());
    while (true) {
        Node next = node;
        for (Node n : node.expand()) {
            if (p.value(n) >= p.value(next)) {
                next = n;
            }
        }
        if (next == node) {
            return node.getState();
        } else {
            node = next;
        }
    }
}
Hill-climbing Search

- Move through state space in the direction of increasing value ("uphill")
Hill-climbing Search

- Move through state space in the direction of increasing value (“uphill”)

State-space landscape
Hill-climbing Search

- Move through state space in the direction of increasing value ("uphill")
State: \([r_0, \ldots, r_7]\)
State: $[r_0, \ldots, r_7]$   Action: $< \hat{i}, r_i >$
State: \([r_0, \ldots, r_7]\)  
Action: \(<i, r_i>\)  
\(h(n) = \# \text{ of pairs of queens attacking each other}\)
\[ h(n) = 17 \]
$h(n) = 17$
$$h(n) = 17$$
\[ h(n) = 17 \]
\[ h(n) = 12 \]
State hillClimb(Problem p) {
    Node node = new Node(p.getInitialState());
    while (true) {
        Node next = null;
        for (Node n : node.expand()) {
            if (p.value(n) >= p.value(node)) {
                next = n;
            }
        }
        if (next == null) {
            return node.getState();
        } else {
            node = next;
        }
    }
}
\( h(n) = 1 \)
objective function

shoulder

global maximum

local maximum

“flat” local maximum

current state

state space
If at first you don’t succeed, try again.
State randomRestart(Problem p) {
    while (true) {
        p.setInitialState(new random State);
        State solution = hillClimb(p);
        if (p.isGoal(solution)) {
            return solution;
        }
    }
}
State randomRestart(Problem p) {
    while (true) {
        p.setInitialState(new random State);
        State solution = hillClimb(p);
        if (p.isGoal(solution)) {
            return solution;
        }
    }
}

Does it work?
State randomRestart(Problem p) {
    while (true) {
        p.setInitialState(new random State);
        State solution = hillClimb(p);
        if (p.isGoal(solution)) {
            return solution;
        }
    }
}

Does it work?  Yes (but)
State randomRestart(Problem p) {
    while (true) {
        p.setInitialState(new random State);
        State solution = hillClimb(p);
        if (p.isGoal(solution)) {
            return solution;
        }
    }
}

Does it work? Yes (but)
How well does it work?
State randomRestart(Problem p) {
    while (true) {
        p.setInitialState(new random State);
        State solution = hillClimb(p);
        if (p.isGoal(solution)) {
            return solution;
        }
    }
}

Does it work? Yes (but)

How well does it work?

Prob of success = $p$  
Expected # of tries = $1/p$
State randomRestart(Problem p) {
    while (true) {
        p.setInitialState(new random State);
        State solution = hillClimb(p);
        if (p.isGoal(solution)) {
            return solution;
        }
    }
}

Does it work?  Yes (but)

How well does it work?

Prob of success = \( p \)  \quad \text{Expected \# of tries} = \frac{1}{p}

= 0.14  \quad \approx 7
State randomRestart(Problem p) {
    while (true) {
        p.setInitialState(new random State);
        State solution = hillClimb(p);
        if (p.isGoal(solution)) {
            return solution;
        }
    }
}
State randomRestart(Problem p) {
    while (true) {
        p.setInitialState(new random State);
        State solution = hillClimb(p);
        if (p.isGoal(solution)) {
            return solution;
        }
    }
}
Randomness in Search

Pure random walk
Complete, but horribly slow

Greedy local search
Incomplete, but fast
Marble Madness
Amiga Version by:
Larry Reed

Copyright (c) 1984, 1986
Atari Games Corp. & Electronic Arts
Simulated Annealing

- Follow landscape down towards global minimum of state cost function
- Occasionally allow an upward move ("shake") to get out of local minima
- Don’t shake so hard that you bounce out of global minimum
Annealing

• A heat treatment that alters the microstructure of a material causing changes in properties such as strength and hardness and ductility

• High temp => Atoms jumping around

• Low temp => Atoms settle into position
State simulatedAnnealing(Problem p, Schedule schedule) {
    Node node = new Node(p.getInitialState());
    for (t=1; true; t++) {
        Number T = schedule(t);
        if (T == 0) {
            return node;
        }
        Node next = randomly selected successor of node
        Number deltaE = p.cost(node) - p.cost(next);
        if (deltaE > 0 || Math.exp(-deltaE/T) > new Random(1)) {
            node = next;
        }
    }
}
\[ y = e^{-\Delta E/10} \]
Simulated Annealing
Simulated Annealing

Complete?
Simulated Annealing

Complete?  Yes, but.
Simulated Annealing

Complete?  Yes, but.

Optimal?
Simulated Annealing

Complete? Yes, but.

Optimal? Yes, but.
Simulated Annealing

Complete?  Yes, but.
Optimal?  Yes, but.

“If the schedule lowers T slowly enough, and the search space is connected, simulated annealing will find a global minimum with probability approaching one.”
Local Search

• Evaluates and modifies a small number of current states

• Does not record history of search

Good: Very little (constant) memory

Bad: May not explore all alternatives

=> Incomplete
Local Beam Search

- During hill-climbing:
  - Keep track of $k$ states rather than just one
  - At each step, generate all successors of all $k$ states ($k^b$ of them)
  - Keep the most promising $k$ of them
Local Search
Local Search
Local Search
Local Search
Local Search
Local Search
Local Search
Parallel Local Search
Parallel Local Search
Parallel Local Search

-8 -5 -7 -2

4 6 9 1

-3 3 0 -9
Local Beam Search
Local Beam Search
Local Beam Search
Local Beam Search
Local Beam Search
Local Beam Search
Local Beam Search
Local Beam Search
Local Beam Search
"And this report just in. ... Apparently, the grass is greener on the other side."
Local Beam Search

• During hill-climbing:
  • Keep track of $k$ states rather than just one
  • At each step, generate all successors of all $k$ states ($k \times b$ of them)
  • Keep the most promising $k$ of them
Asexual Reproduction
Mitosis

DNA replication

Mitosis

Two diploid cells
Sexual Reproduction
Meiosis

Meiosis

Interphase

Meiosis I

Homologous Chromosomes

Meiosis II

Daughter Nuclei

Nuclei II

Wikipedia
Genetic Algorithms

• Start with $k$ random states
• Select pairs of states and have them “mate” to produce “offspring”
• Most fit (highest-scoring) individuals reproduce more often
Genetic Algorithms

- States encoded as “chromosomes” (linear sequences, a.k.a. strings)
- During mating:
  - Crossover: swap chunks of code
  - Mutation: randomly change bits of code
Initial Population

Fitness Function

Selection

Crossover

Mutation
Genetic Algorithms

- States encoded as “chromosomes” (linear sequences, a.k.a. strings)

- During mating:
  - Crossover: swap chunks of code
  - Mutation: randomly change bits of code
GA Summary

• A version of stochastic local beam search with a special way to generate successor states (motivated by a naive biology analogy)

• “Much work remains to be done to identify the conditions under which genetic algorithms perform well.”
Hill-climbing

Local Beam Search

Simulated Annealing

Genetic Algorithms
• Evaluates and modifies a small number of current states

• Does not record history of search

**Hill-climbing**

**Good:** Very little (constant) memory

**Bad:** May not explore all alternatives

=> Incomplete
For next time:
AIMA 5.0–5.2.2

Upper-level writing: Topics due:
What subject will you be writing about?
What questions about it will you try to answer?

Homework 1: Do it (really!)