CSC242: Artificial Intelligence

Lecture 2: Problem Solving
One More Policy
One More Policy
General Problem Solving
\[ |\vec{F}_{21}| = |\vec{F}_{12}| = \frac{G M_1 M_2}{d^2} \]
Cooperative Problem Solving
Cooperative Problem Solving
Cooperative Problem Solving
We don’t make the computers.*
We make the computers solve problems.

*Or the programming languages, compilers, debuggers, databases, graphics pipelines, network protocols, web servers, ...
Our first problem
Given start city A and destination city B:

Can I get from A to B?
Given start city A and destination city B:

Can I get from A to B?
How do I get from A to B?
Given start city A and destination city B:

Can I get from A to B?

How do I get from A to B?

Minimize tolls

Deadline

Stay close to rest areas

Mileage limit
Cities

Roads: connect cities

Romania
Cities

Roads: connect cities

Distances between cities

Romania

Cities

Distances between cities

Roads: connect cities

Romania
What problem are we trying to solve?
What problem are we trying to solve?

What aspects of the world are important to solving that problem?
Problem Solving by Computers
Given initial city $src$ and destination city $dst$, find a route from $src$ to $dst$ (if one exists).
Given initial city src and destination city dst, find a route from src to dst (if one exists).

**Approach #1**

Precompute, for each <src,dst> pair, the shortest route between them. Then, to get from src to dst, just lookup the stored route.
Given initial city $src$ and destination city $dst$, find a route from $src$ to $dst$ (if one exists).

**Approach #1**

Precompute, for each $<src,dst>$ pair, the shortest route between them. Then, to get from $src$ to $dst$, just lookup the stored route.

Does it work?
Given initial city src and destination city dst, find a route from src to dst (if one exists).

**Approach #1**
Precompute, for each <src,dst> pair, the shortest route between them. Then, to get from src to dst, just lookup the stored route.

Does it work? Yes
Given initial city \textit{src} and destination city \textit{dst}, find a route from \textit{src} to \textit{dst} (if one exists).

\textbf{Approach \#1}
Precompute, for each \textit{<src,dst>} pair, the shortest route between them. Then, to get from \textit{src} to \textit{dst}, just lookup the stored route.

\textbf{Does it work?} Yes

\textbf{Problems?}
Given initial city src and destination city dst, find a route from src to dst (if one exists).

Approach #1
Precompute, for each <src,dst> pair, the shortest route between them. Then, to get from src to dst, just lookup the stored route.

Does it work?  Yes

Problems?
• Lookup table grows as $O(n^2)$
Given initial city \( src \) and destination city \( dst \), find a route from \( src \) to \( dst \) (if one exists).

**Approach #1**

Precompute, for each \(<src, dst>\) pair, the shortest route between them. Then, to get from \( src \) to \( dst \), just lookup the stored route.

**Does it work?** Yes

**Problems?**

- Lookup table grows as \( O(n^2) \)
- Computing all-pairs shortest path: \( O(n^3) \)
Given initial city src and destination city dst, find a route from src to dst (if one exists).

Approach #1
Precompute, for each <src, dst> pair, the shortest route between them. Then, to get from src to dst, just lookup the stored route.

Does it work? Yes

Problems:
• Lookup table grows as $O(n^2)$
• Computing all-pairs shortest path: $O(n^3)$
• Changes to network: recompute entire table
Given initial city \textit{src} and destination city \textit{dst}, find a route from \textit{src} to \textit{dst} (if one exists).

\textbf{Approach #2}
Given initial city src and destination city dst, find a route from src to dst (if one exists).

Approach #2
If I were at src, and I wanted to get to dst, what could I do?
Given initial city src and destination city dst, find a route from src to dst (if one exists).

Approach #2
If I were at src, and I wanted to get to dst, what could I do?

➡ Move to any adjacent city
Given initial city src and destination city dst, find a route from src to dst (if one exists).

Approach #2
If I were at src, and I wanted to get to dst, what could I do?

➡ Pick an adjacent city
Move to it
Given initial city $src$ and destination city $dst$, find a route from $src$ to $dst$ (if one exists).

Approach #2
If I were at $src$, and I wanted to get to $dst$, what could I do?

➡️ Pick an adjacent city
Move to it
Until I’m at $dst$
find_solution(City src, City dst) {
    City[] solution = [];
    City c = src;
    while (c != dst) {
        City[] neighbors = adjacent_cities(c);
        City next_c = select_one(neighbors);
        solution.add(next_c);
        c = next_c;
    }
    return solution;
}
find_solution(City src, City dst) {
    City[] solution = [];
    City c = src;
    while (c != dst) {
        City[] neighbors = adjacent_cities(c);
        City next_c = select_one(neighbors);
        solution.add(next_c);
        c = next_c;
    }
    return solution;
}
Intelligence and Generality

- Intelligence includes the ability to solve many kinds of problems
- Including problems we haven’t seen before
- Every new problem-solving method needs to be designed, implemented, tested, and debugged
Given initial puzzle configuration \textit{start} and desired configuration \textit{goal}, find a sequence of moves that goes from \textit{start} to \textit{goal} (if one exists).
State
(Data Structure)
find_solution(City src, City dst) {
    City[] solution = [];
    City c = src;
    while (c != dst) {
        City[] neighbors = adjacent_cities(c);
        City next_c = select_one(neighbors);
        solution.add(next_c);
        c = next_c;
    }
    return solution;
}
State
(Data Structure)

<table>
<thead>
<tr>
<th>7</th>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
State
(Data Structure)

$$M_{i,j} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix}$$
State (Data Structure)

\[
M_{i,j} = \begin{bmatrix}
7 & 0 & 1 \\
6 & 2 & 8 \\
3 & 4 & 5 \\
\end{bmatrix}
\]

```java
int[][] M = new int[3][3];
M[0][0] = 7;
M[0][1] = 0;
... 
M[2][2] = 5;
```
Actions

• Can be performed in a state
• Change the state to a resulting state
find_solution(City src, City dst) {
    City[] solution = [];
    City c = src;
    while (c != dst) {
        City[] neighbors = adjacent_cities(c);
        City next_c = select_one(neighbors);
        solution.add(next_c);
        c = next_c;
    }
    return solution;
}
<table>
<thead>
<tr>
<th>7</th>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Path:
Path: 1

<table>
<thead>
<tr>
<th>7</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Path: 1 - 8 - 2
Path: 1 - 8 - 2 - 6
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Path: East
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Path: East - South
Path: East - South - West -
Path: East - South - West - West
Actions

• For any state and action:
  • Can I perform this action in this state?
  • “Applicability”
  • How do I update the state if this action is performed?
  • “Result” or “transition” function
\[ M^{\text{in}} = \begin{bmatrix}
7 & 0 & 1 \\
6 & 2 & 8 \\
3 & 4 & 5 \\
\end{bmatrix} \quad M^{G} = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 0 \\
\end{bmatrix} \]
$$M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$
\[
M^{in} = \begin{bmatrix}
7 & 0 & 1 \\
6 & 2 & 8 \\
3 & 4 & 5
\end{bmatrix} \quad M^G = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 0
\end{bmatrix}
\]

Approach #1
Precompute, for each \(<M^{in}, M^G>\) pair, a sequence of moves that transforms \(M^{in}\) into \(M^G\). Then, to solve \(M^{in}\), just lookup the stored sequence.
$M_{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^G = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$

Approach #1
Precompute, for each $<M_{\text{in}}, M^G>$ pair, a sequence of moves that transforms $M_{\text{in}}$ into $M^G$. Then, to solve $M_{\text{in}}$, just lookup the stored sequence.

Does it work?
$M^{in} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^G = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$

Approach #1
Precompute, for each $<M^{in},M^G>$ pair, a sequence of moves that transforms $M^{in}$ into $M^G$. Then, to solve $M^{in}$, just lookup the stored sequence.

Does it work? No
\[ M^{in} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^{G} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \]

Approach #1

Precompute, for each \( <M^{in},M^{G}> \) pair, a sequence of moves that transforms \( M^{in} \) into \( M^{G} \). Then, to solve \( M^{in} \), just lookup the stored sequence.

Does it work? No

Problems?
$M^{in} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^{G} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$

**Approach #1**

Precompute, for each $<M^{in}, M^{G}>$ pair, a sequence of moves that transforms $M^{in}$ into $M^{G}$. Then, to solve $M^{in}$, just lookup the stored sequence.

**Does it work?** No

**Problems?**

- There are $9! = 362880$ ($O(n!)$ in general) cases
\[
M^{in} = \begin{bmatrix}
7 & 0 & 1 \\
6 & 2 & 8 \\
3 & 4 & 5
\end{bmatrix} \quad M^{G} = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 0
\end{bmatrix}
\]

Approach #1
Precompute, for each \( <M^{in},M^{G}> \) pair, a sequence of moves that transforms \( M^{in} \) into \( M^{G} \). Then, to solve \( M^{in} \), just lookup the stored sequence.

Does it work? No

Problems?

- There are \( 9! = 362880 \) (\( O(n!) \) in general) cases
- No obvious way to solve the \( <M^{in},M^{G}> \) cases
Approach #1
Precompute, for each $<M^{in}, M^{G}>$ pair, a sequence of moves that transforms $M^{in}$ into $M^{G}$. Then, to solve $M^{in}$, just lookup the stored sequence.

Does it work? No

Problems?
• There are $9! = 362880$ (O(n!) in general) cases
• No obvious way to solve the $<M^{in}, M^{G}>$ cases
• This problem is known to be NP-complete
\[ M^{\text{in}} = \begin{bmatrix} 7 & 0 & 1 \\ 6 & 2 & 8 \\ 3 & 4 & 5 \end{bmatrix} \quad M^{\text{G}} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix} \]

Approach #2
Approach #2

If the puzzle is currently $M$, and I want it to be $M^G$, what could I do?

\[
M^\text{in} = \begin{bmatrix}
7 & 0 & 1 \\
6 & 2 & 8 \\
3 & 4 & 5
\end{bmatrix} \quad M^G = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 0
\end{bmatrix}
\]
Approach #2

If the puzzle is currently $M$, and I want it to be $M^G$, what could I do?

→ Move the blank to an adjacent space
Approach #2
If the puzzle is currently $M$, and I want it to be $M^G$, what could I do?

➡ Pick an adjacent space
  Move the blank to it
If the puzzle is currently $M$, and I want it to be $M^G$, what could I do?

- Pick an adjacent space
  - Move the blank to it

Until $M == M^G$
Given initial city src and destination city dst, find a route from src to dst (if one exists).

Approach #2
If I were at src, and I wanted to get to dst, what could I do?

➡ Pick an adjacent city
    Move to it

Until I’m at dst
Initial state: $S^{IN}$  
Goal state: $S^{G}$

If the problem state is currently $S$, and I want it to be $S^{G}$, what could I do?

➡ Pick an applicable action $A$
   Update $S$ with the result of applying $A$

Until $S == S^{G}$
find_solution(State initial) {
    State s = initial;
    Action[] solution = [];
    while (!is_goal(s)) {
        Action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(State initial) {
    State s = initial;
    Action[] solution = [];
    while (!is_goal(s)) {
        Action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(State initial) {
    State s = initial;
    Action[] solution = [];
    while (!is_goal(s)) {
        Action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(State initial) {
    State s = initial;
    Action[] solution = [];
    while (!is_goal(s)) {
        Action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(State initial) {
    State s = initial;
    Action[] solution = [];
    while (!is_goal(s)) {
        Action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(State initial) {
    State s = initial;
    Action[] solution = [];
    while (!is_goal(s)) {
        Action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(State initial) { 
    State s = initial;
    Action[] solution = [];
    while (!is_goal(s)) {
        Action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(State initial) {
    State s = initial;
    Action[] solution = [];
    while (!is_goal(s)) {
        Action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(State initial) {
    State s = initial;
    Action[] solution = [];
    while (!is_goal(s)) {
        Action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
Problem (Domain): \( \langle S, A, \text{Actions}, \text{Result} \rangle \)

**Actions**: \( s \in S \rightarrow \{a \in A : a \text{ can be executed (is applicable) in } s \} \)

**Result**: \( s \in S, a \in A \rightarrow \)

\( s' : s' \in S \text{ s.t. } s' \text{ is the result of performing } a \text{ in } s \)

Problem (Instance): \( \langle I \in S, G \subseteq S \rangle \)

Solution: \( \langle a_0, a_1, \ldots, a_n \rangle \in A \text{ s.t.} \)

\( \text{Result}(\cdots \text{Result}(\text{Result}(I, a_0), a_1) \cdots, a_n) \in G \)
World state
World state
Problem-solving state

World state
Problem-solving state

World state
Action
Problem-solving state
Transition Model

World state
Action
find_solution(initial) {
    state s = initial;
    action[] solution = [];
    while (!is_goal(s)) {
        action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(initial) {
    state s = initial;
    action[] solution = [];
    while (!is_goal(s)) {
        action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(initial) {
    state s = initial;
    action[] solution = [];
    while (!is_goal(s)) {
        action a = pick(actions(s));
        solution.add(a);
        s = result(s, a);
    }
    return solution;
}
find_solution(initial) {  
    state s = initial;  
    action[] solution = [];  
    while (!is_goal(s)) {  
        action a = pick(actions(s));  
        solution.add(a);  
        s = result(s, a);  
    }  
    return solution;  
}
State-Space Search
States + Actions + Transition Model

= State Space

The set of all states reachable from the initial state by some sequence of actions
State Space

\[ \langle V, E \rangle : \]

\[ V = \{ v_i \mid s_i \in S \} \]

\[ E = \{ \langle v_i, v_j, a \rangle \mid s_j = \text{RESULT}(s_i, a) \} \]
State-Space Search

- Start with initial state
- Generate successor states by applying applicable actions
- Until you find a goal state
Solution `treeSearch`(Problem p) {
    Set<Node> frontier = new Set<Node>(p.getInitialState());

    while (true) {
        if (frontier.isEmpty()) {
            return null;
        }
        Node node = frontier.selectOne();
        if (p.isGoalState(node.getState())) {
            return n.getSolution();
        }

        for (Node n : node.expand()) {
            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 n.getSolution();
        }
        explored.add(node);
        for (Node n : node.expand()) {
            if (!explored.contains(n)) {
                frontier.add(n);
            }
        }
    }
}
Summary

• General-purpose algorithm for solving any problem that can be represented using states and actions that transition between them

• State-space search framework will allow us to explore and compare alternatives
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);
            }
        }
    }
}
Your Homework this Weekend:
Read Chapters 2 and 3