

Massachusetts Institute of Technology

16.410-13 Principles of Autonomy and Decision Making

Assignment #3

Due: Session 8

Objective

To walk through the mechanics of performing uninformed and informed search. To develop an understanding of what constitutes an admissible heuristic. To analyze and compare the worst case time and space performance of several search algorithms.

Building upon the code that you developed for problem set 2, you will implement in Java the following **informed** search algorithms:

1. A*
2. Branch and Bound

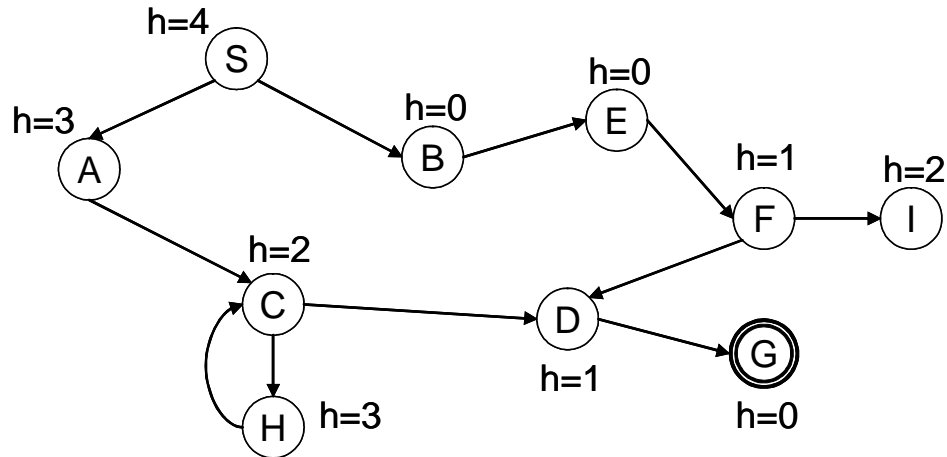
You will use the implemented search algorithms to solve several problems. You will run your algorithms on a sequence of five different graphs and analyze the results.

Background

Lecture notes L6 and L7, “Exploring Roadmaps using Uninformed Search” and “Formulating Path Planning using Roadmaps”. Also, please read about informed search and exploration in AIMA, Ch. 4.1-2.

Problem 1: Search

You are trying to find a path from the state **S** to state **G** in the following directed graph using several search algorithms.



Assume the following:

- Edges are length 1.
- H denotes heuristic cost for each node.
- Given two or more equally good nodes, explore them in **alphabetical order**.
- The search stops as soon as the goal is expanded.
- Each search algorithm uses a **visited list** to avoid revisiting expanded nodes.

Write the sequence of nodes **expanded** by the specified search methods. A node is **expanded** when the method takes it off the queue, and attempts to create its children.

Part A Iterative Deepening

Show the iterative-deepening expansion sequence up until the first solution path has been found (we have started it for you). Please note that the visited list is cleared between iterations of iterative deepening.

Iteration Depth: **Expansion Sequence:**

1	S
2	S-A-B
3	S-...

For Greedy Search, give the final path found. Is this the optimal solution?

S-...

Part B Greedy Search

Show the greedy search expansion sequence up until the first solution path has been found (we have started it for you). **Write above each node** in your expansion sequence **the number used by greedy search** to pick the next node to expand.

0 1 ...
S-A-...

For Greedy Search, give the final path found. Is this the optimal solution?

S-...

Part C A* Search

Show the A* expansion sequence up until the first solution path has been found (we have started it for you). **Write above each node** in your expansion sequence **the number used by A*** to pick the next node to expand. For A* search, please show your work, including queue and visited list.

4 0 ...
S-B-...

For A* search, give the final path found. Is this the optimal solution?

S-...

Part D Branch and Bound Search

Show the Branch and Bound expansion sequence up until the first solution path has been found (we have started it for you). For Branch and Bound search, please draw the search tree, and indicate each new incumbent solution that is found by putting a box around the goal node, along with the cost of the new incumbent solution. Also, indicate any pruned nodes by putting an X across the node.

S-A-...

For Branch and Bound search, give the final path found. Is this the optimal solution?

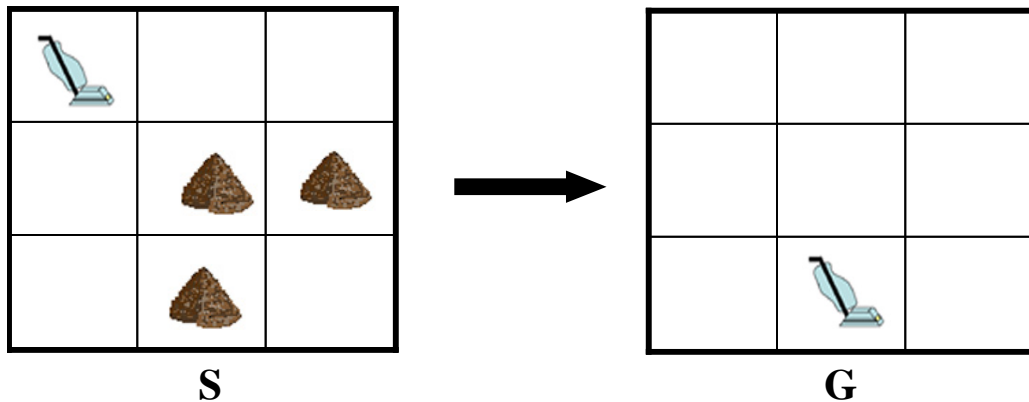
S-...

Problem 2: An Admissible Heuristic

For Problem 2, we revisit the vacuum world in Problem Set 1. However, this time there are nine rooms in a 3x3 grid pattern. Your job is to develop one admissible heuristic and one inadmissible heuristic to improve the speed of cleaning the vacuum world.

Assume the following:

- the vacuum cleaner can move between any two rooms that share a common wall
- we are using State Encoding 1 (it is listed below for convenience)
- only one operator can be applied at each search step
- the initial state, S , is indicated below
- the vacuum cleaner is initially turned off and in the top left corner
- each initially dirty room has a pile of dirt in it, each initially empty room is clean
- the goal is to have all rooms clean
- it doesn't matter which room the vacuum cleaner is in when the goal is achieved



Remember, a heuristic function should measure the **difference** between two states of the vacuum world, and that an admissible heuristic should be an underestimate of the number of moves to the goal.

Part A Admissible Heuristic

For the vacuum world, develop one admissible heuristic to aid in solving the problem, and explain why your heuristic is admissible. (A one paragraph answer is sufficient)

Part B Inadmissible Heuristic

For the vacuum world, also develop one sensible, but inadmissible heuristic. Explain why this heuristic is inadmissible. (A one paragraph answer is sufficient)

Part C Estimating Distance to the Goal

Using your admissible heuristic from Part A, determine a lower bound on the number of steps to the goal configuration from the configuration S depicted above. Is the goal configuration actually achievable in this number of steps? Why, or why not? (A one paragraph answer is sufficient)

State Encoding 1 is repeated below for convenience:

```
Left:    {dirty, empty}
Right:   {dirty, empty}
Vacuum:  {left, right}
Suction: {on, off}
```

For State Encoding 1, we encode the four operators as follows:

```
move-left:
    <?, ?, right, off> -> <    ?, ?, left, off>
    <?, ?, right, on> -> <empty, ?, left, on>

move-right:
    <?, ?, left, off> -> <?,           ?, right, off>
    <?, ?, left, on> -> <?, empty, right, on>

turn-on:
    <?, ?, left, off> -> <empty,     ?, left, on>
    <?, ?, right, off> -> <    ?, empty, right, on>

turn-off:
    <?, ?, ?, on> -> <?, ?, ?, off>
```

Problem 3: Performance Analysis

Part A Performance

In answering these questions, assume that the search space is a tree with branching factor b , and depth d . Note that the number of nodes in such a tree is $(b^{(d+1)}-1)/(b-1)$. Since we only care about the order of growth, we will abbreviate this to $b^{(d+1)}$. The running time is proportional to the number of nodes visited and the space is proportional to the maximum length of the Q in our simple implementation.

1. What is the (approximate) worst case running time for hill-climbing search (without backup)?
2. What is the (approximate) worst case space for hill-climbing search (without backup)?
3. What is the (approximate) worst case running time for hill-climbing search (with backup)?
4. What is the (approximate) worst case space for hill-climbing search (with backup)?
5. What is the (approximate) worst case running time for best-first search?
6. What is the (approximate) worst case space for best-first search?
7. What is the (approximate) worst case running time for beam search (with beam-width = k)?
8. What is the (approximate) worst case space for beam search (with beam-width = k)?

Part B A* Search Characteristics

Indicate which of the following statements are true.

1. A* is guaranteed to find an optimal solution (assume that a solution does exist).
2. A solution found by A* is guaranteed to be optimal (assume an inadmissible heuristic is used).
3. The admissible heuristic in A* must never over-estimate the true cost of a solution.
4. In a search of a square grid, allowed moves are North, South, East, and West. The cost of such a move is the distance traveled (1 mile). All nodes exist at intersections of the grid. The Euclidean distance (square root of vertical and horizontal distance) is not an appropriate A* heuristic.

Problem 4 Implementation

Problems 4, 5, and 6 extend your implementation from Problem Set 2 by adding support for informed search and analyzing the performance empirically.

Add to the general search capability, that you developed in problem set 2, methods that implement A*-search and Branch-and-Bound-search. You will need to add support for providing a heuristic method.

Add `JUnit` tests for each of these (two) new search algorithms. Demonstrate that your implementations of the two new search algorithms work correctly and handle exceptional cases such a mal-formed data structures or failed searches gracefully.

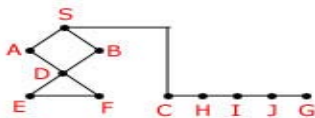
```
class AStarSearch implements GeneralizedSearch
{ /* Implement this class */ }
class BranchAndBoundSearch implements GeneralizedSearch
{ /* Implement this class */ }
```

If `GeneralizedSearch` is an abstract class in your implementation then your classes should extend `GeneralizedSearch`.

You should still be able to run all of the algorithms from Problem Set 2 as well as the two new algorithms.

Problem 5 Demonstrating A* and BnB on a simple example

In this problem we extend the simple example given in Problem Set 2 by adding two kinds of information. To the vertices we add an XY position (`XYLocation` interface). And for the edges we give an edge length (`RoadLength` interface). As before, we are trying to find a path from S to G in the following graph:



The graph has been provided in XML format (`ps3data.xml`) and a reader of the file has been provided.

The XML file for the above graph looks like this:

```
<GRAPH>
  <VERTEX name="S" x="2" y="6">
    <EDGE vertex="A" length="4"> </EDGE>
    <EDGE vertex="B" length="4"> </EDGE>
```

```

    <EDGE vertex="C" length="12"> </EDGE>
  </VERTEX>
<VERTEX name="A" x="0" y="4">
  <EDGE vertex="S" length="4"> </EDGE>
  <EDGE vertex="D" length="4"> </EDGE>
</VERTEX>
<VERTEX name="B" x="4" y="4">
  <EDGE vertex="S" length="4"> </EDGE>
  <EDGE vertex="D" length="4"> </EDGE>
</VERTEX>
<VERTEX name="C" x="8" y="0">
  <EDGE vertex="S" length="12"> </EDGE>
  <EDGE vertex="H" length="4"> </EDGE>
</VERTEX>
<VERTEX name="D" x="2" y="2">
  <EDGE vertex="A" length="4"> </EDGE>
  <EDGE vertex="B" length="4"> </EDGE>
  <EDGE vertex="E" length="4"> </EDGE>
  <EDGE vertex="F" length="4"> </EDGE>
</VERTEX>
<VERTEX name="E" x="0" y="0">
  <EDGE vertex="D" length="4"> </EDGE>
  <EDGE vertex="F" length="4"> </EDGE>
</VERTEX>
<VERTEX name="F" x="4" y="0">
  <EDGE vertex="D" length="4"> </EDGE>
  <EDGE vertex="E" length="4"> </EDGE>
</VERTEX>
<VERTEX name="G" x="24" y="0">
  <EDGE vertex="J" length="4"> </EDGE>
</VERTEX>
<VERTEX name="H" x="12" y="0">
  <EDGE vertex="C" length="4"> </EDGE>
  <EDGE vertex="I" length="4"> </EDGE>
</VERTEX>
<VERTEX name="I" x="16" y="0">
  <EDGE vertex="H" length="4"> </EDGE>
  <EDGE vertex="J" length="4"> </EDGE>
</VERTEX>
<VERTEX name="J" x="20" y="0">
  <EDGE vertex="I" length="4"> </EDGE>
  <EDGE vertex="G" length="4"> </EDGE>
</VERTEX>
</GRAPH>

```

As you can see, the XML consists of one <GRAPH> ... </GRAPH> which contains all of the vertices (<VERTEX></VERTEX>) each of which contains edges to other names

vertices. This is identical to the ps2data.xml file provided in Problem Set 2 except that the vertices have XY location information and the edges have edge length information consistent with the drawing of the graph given above.

We have provided Java classes for the vertices and edges that extend the ones given in Problem Set 2 as follows:

```

XYLocation.java
public interface XYLocation {
    double getXlocation ();
    double getYlocation ();
    public void setXlocation (double x);
    public void setYlocation (double y);
}

RoadLength.java
public interface RoadLength
{
    double getRoadLength ();
}

Ps3Vertex.java
import java.util.LinkedList;
import java.util.List;

public class Ps3Vertex extends Ps2Vertex implements XYLocation
{
    double x;
    double y;

    Ps3Vertex (String vertexname) { super(vertexname); x=0.0; y=0.0; }
    Ps3Vertex (String vertexname, double x, double y) {
        super(vertexname);
        this.x=x;
        this.y=y;
    }
    public double getXlocation () { return x; }
    public double getYlocation () { return y; }
    public void setXlocation (double x) { this.x=x; }
    public void setYlocation (double y) { this.y=y; }
}

Ps3Edge.java

public class Ps3Edge extends Ps2Edge implements RoadLength
{
    double roadlength;
    Ps3Edge (Ps2Vertex vertex) { super(vertex); }
    Ps3Edge (Ps2Vertex vertex, double rl) { super(vertex); roadlength=rl; }
    public double getRoadLength () { return roadlength; }
}

```

The provided XML reading code Ps3XMLGraph.java which, as before, can be used to read in the Graph data as follows:

```

// Read in the graph
Ps3XMLGraph xmlreader=new Ps3XMLGraph();
Dictionary vertices=xmlreader.readGraph("ps3data.xml");
// Find the start and end vertices
startVertex=(Ps3Vertex)vertices.get("S");
endVertex=(Ps3Vertex)vertices.get("G");

```

Run your four algorithms on this (ps3data.xml) data and report your results.

Problem 6 Empirical Evaluation

We have provided a collection of graphs as XML files in files gs1.xml, gs2.xml, gs3.xml, gs4.xml, and gs5.xml. In each case search from node “S” to node “G” and analyze the metric information from each run using graphs to aid in your analysis.

Construct an example for each algorithm in which it would perform poorly compared to the others.

Compare the relative performance of BFS, DFS, IDA, A*, and Branch and Bound in terms of memory and space as a function of graph size and connectivity.

Comment on why each algorithm performs well or poorly.

Problem 7 Time

Please let us know the amount of time it took you to complete this problem set.