**COMP 8700 Fall 2020 Final Project**

**Darshil M. Bhakhar**,

Master of Applied Computing,

University of Windsor,

bhakhar@uwindsor.ca

**Gurneet Singh**,

Master of Applied Computing,

University of Windsor,

gurneet@uwindsor.ca

**Dharmikkumar V. Patel**,

Master of Applied Computing,

University of Windsor,

patel8m2@uwindsor.ca

**Dharmikkumar V. Patel**,

Master of Applied Computing,

University of Windsor,

patel8m2@uwindsor.ca

**Wen Dong**,

Master of Applied Computing,

University of Windsor,

dong23@uwindsor.ca

***Abstract –*** this is the report of our team final project of Artificial Intelligence Introduction, in this project, we have experimented and explored a set of search algorithms including informed and uninformed ones with the famous Eight Puzzles Game, and found that the performance of different algorithms varies markedly in terms of optimality, completeness, time complexity and space complexity; we also found that heuristic function plays a significant role in the search for informed algorithms, and impact the outcome tremendously.

**Introduction**

The project is the final project of our AI Introduction course, which is to put the comprehensive knowledge and theory we have learned from class into reality, from abstract intuition to tangible experiment. It is beneficial for all the team members to consolidate the learning and build hands-on experience on Artificial Intelligence.

In the project, we have chosen and explored a set of informed algorithms:

* AStarSearch – A graph traversal and path search algorithm, an import and widely used search algorithm in computer science due to its completeness, optimality and optimal efficiency. It uses function f(n) = g(n) + h(n) as estimate function, based on which to expand cheapest node first. The sole drawback is its space complexity therefore not practical for large-scale problems.
* RecursiveBestFirstSearch – It is a variant version of A Star for large-scale problems where A Star is unable to tackle due to space complexity.
* GreedyBestFirstSearch – It is designed to find goal quickly by expanding the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly with evaluate function f(n) = h(n). So, nodes not on the solution path are not expanded. Cost low, however, it is not optimal neither complete because optimal solution might be in other unexplored paths.

Additionally, two uninformed as well:

* BreadthFirstSearch – Breadth-First is a search strategy in which the nodes are expanded and explored in a breadth first way, that is the root node expanded, then the successors are expanded and so on. When all steps costs are equal, it is optimal as it always expands the shallowest unexpanded node.
* UniformCostSearch – with a simple extension on Breadth-First search, Uniform-Cost Search is optimal for any step-cost function. Instead of expanding the shallowest node, it expands the node with lowest path cost g(n)

Aside from the variety of algorithms, we have also employed three different heuristic functions and two different accessory strategies, i.e. tree search and graph search to work with the informed searches, to observe how they impact the searching process.

**Experiment**

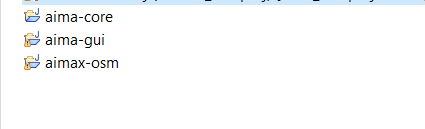
Our experiment is based on the open source project AIMA which is the java implementation of algorithms from Russell and Norvig’s Artificial Intelligence – A Modern Approach 3rd Edition.

Setup

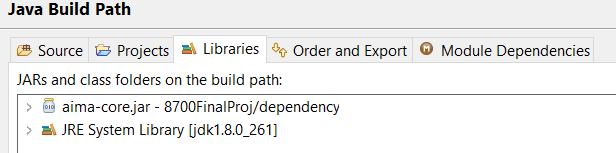
1. Download the latest source code of AIMA java from the GitHub repository

<https://github.com/aimacode/aima-java>

1. open the projects with Eclipse (Version: 2020-09 (4.17.0)) with builder Java 8+



1. Export aima-core as a jar library then add and reference it in our project



1. Finally, we are ready to import all sorts of algorithms and eight puzzles related assets from the library to build our testing flows of interest.

Coding

Based on the AIMA project, coding the experimental flows is like a breeze, as most of the work can be achieved simply by permutation and combination of various existential java classes. For instances, testing A Star with graph search and Manhattan heuristic function is implemented as below:

1. Prepare initial board states to search solution for
2. Create an instance of Manhattan heuristic function class

ManhattanHeuristicFunction hf = new ManhattanHeuristicFunction();

1. Create an instance of A Star search algorithm with the heuristic function and a graph search
2. Assembly the objects together to create a Problem

Problem problem = **new** Problem(initialState,

EightPuzzleFunctionFactory.*getActionsFunction*(),

EightPuzzleFunctionFactory.*getResultFunction*(),

**new** EightPuzzleGoalTest());

1. Instantiate a Search Agent to conduct the actual search, and output the search result in the end.

SearchAgent agent = new SearchAgent(problem, search);

printInstrumentation(agent.getInstrumentation());

**Testing Results and Analysis**

1. Comparison between algorithms

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Algorithm | pathCost | nodesExpanded | queueSize | maxQueueSize |  |
| Informed | A Star | 18 | 227 | 156 | 157 | Manhattan heuristic function and graph search used. With initial state  5 4 0  3 2 8  6 1 7 |
| Recursive Best First | 18 | 472607 | N/A | 18 |
| Greedy Best First | 58 | 309 | 228 | 229 |
| Uninformed | Breadth First | 18 | 13210 | 7396 | 7396 |  |
| Uniform Cost | 18 | 19268 | 11020 | 11023 |  |

From the result, we can see except Greedy Best Frist search, all the others are optimal, among which A Star performs well in the number of expanded nodes, and also with a decent queue size and maximum queue size comparing to uninformed approaches.

1. Comparison between Tree search and Graph search
2. Comparison between heuristic functions

References:

<https://github.com/aimacode/aima-java> AIMA3e branch

[https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)