Study Of AI Algorithms For The Solution Of Puzzles With Minimum Number Of Moves

Luke Garrigan

registration: 100086495

1 Introduction

The goal of this project is to design, implement and compare algorithms for solving puzzles such as the sliding-tile and Rubik's Cube in the minimum number of moves. I will be implementing Dijkstra's single-source shortest-path which is an algorithm used to find shortest paths from one given vertex to all other vertices in a non-negatively weighted graph. An example for this algorithm could be represented in road networks Nordhoff and Lammich (2012).

I will be designing the A* algorithm and analyse its applicability with its exponential growth Korf (1993). The A* algorithm is widely used for path-finding and graph traversal, the course of plotting and efficiently traversing paths between nodes.

The iterative deepening A* algorithm is my main focus, it is also a graph traversal and path search algorithm with the goal to find the shortest path between nodes. IDA* is a depth-first search algorithm thus its memory usage is lower than A*, however it doesn't use dynamic programming thus often ends up exploring the same nodes multiple times. I will be analysing and comparing it with the A* with Recursive Best First Search algorithm. RBFS is a recurisve algorithm in which attempts to mimic the operations of A* search, however it executes using a linear space rather than A*'s exponential space complexity. RBFS uses an evaluation limit to prevent it continually going down a specific path Hatem et al. (2015).

I will also incorporate and test alternative algorithms such as Memory Bounded A*, which is a shortest path algorithm based on A* with the main advantage of using bounded memory while the A* algorithm utilises exponential memory. I will additionally produce my own personal solutions and methods.

I will be implementing the algorithms in Java as I have years of experience java programming, from developing my own games with built in artificial intelligence to path finding using the A* search algorithm for finding the shortest path.

2 The Sliding Tile Puzzle

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Figure 1: The goal states of the Fifteen and Twenty-Four Puzzles

In the Sliding tile Puzzle any tile which is horizontally or vertically adjacent to the blank tile can be repositioned into that empty space. The goal of the puzzle is to move the tiles from a set composition to the goal state as shown in Figure 1. The main objective is to reach the target state in a minimum number of moves.

I will be focusing on the 4x4 and 5x5 sliding puzzles because they require a heuristic search algorithm such as A*, whereas a 3x3 sliding puzzle could be solved with brute force as it only has a total of 181,440 states and can be solved optimally with a breadth-first search Korf and Felner (2002). I will be testing and evaluating different types of heuristics for this given problem starting off with the Manhattan distance. The Manhattan distance is The distance between two points in a grid, based on a strictly horizontal and/or vertical path. The Manhattan distance is the simple sum of the horizontal and vertical components.

I will be testing the linear-conflict heuristic which has a significant improvement over the Manhattan distance, as it considers the goal row or column, but reversed relative to each other. What this means is that two tiles `X` and `Y` are in linear conflict if `X` and `Y` are in the same line as well as the goal positions, `Y` is to the left of `X` and goal position of `Y` is to the right of goal position of `X`. Both the linear-conflict heuristic and Manhattan heuristic are admissible meaning they don't overestimate the cost of reaching the goal.

3 The Rubiks Cube

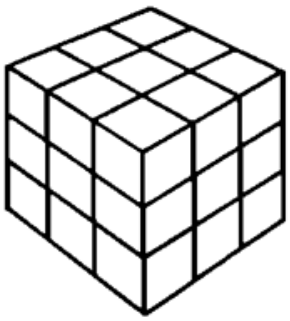

Figure 2: The Rubik's Cube

The Rubik's Cube is the worlds best known combinatorial puzzle. Each 3x3 plane can be rotated 90 or 180 degrees, clockwise or anticlockwise. The goal of the game is to have all 6 sides showing only one colour. The 3x3x3 Rubik's Cube contains about 4.3252x1019 different reachable states.

There are 20 movable subcubes, or cubies, which can be divided into eight corner cubies, with three faces each, and twelve edge cubies, with two faces each. There are 88,179,840 different positions and orientations of the corner cubies, and the number of

moves needed to solve just the corner cubies ranges from zero to eleven.Korf and Felner (2002)

My goal is to implement the IDA* algorithm which will solve the Rubik's cube in a minimum number of moves in the best possible time. Richard E. Korf predicted the Time complexity of iterative-deepening-A* algorithm for the Rubik's cube within 1% accuracy in every case. Korf et al. (2001)

4 Risk Analysis

The main risks which I face in tackling this project are based primarily around the time allocated. I have many algorithms which I want to design, implement and experiment for both the sliding tile puzzle and the Rubik's Cube, that I may run into trouble with time and struggle to finish certain algorithms. I have prepared for this and set out a priority ordered list for the algorithms which I will develop based upon. As shown by the Gantt Chart I have allocated the main algorithms a time slot in which they share with the lesser priority algorithms, this method will allow for completion of higher priority algorithms before beginning development of lower priority ones as shown in Table 1.

Table 1: Algorithm Priority Table

Algorithm	Sliding Tile	Rubiks Cube
Iterative deepening A*	1	1
Single-source shortest-path	3	3
A* with Recursive Best First Search	2	2
Memory Bounded A*	3	3

In developing this prioritisation system I should be able to get the core components necessary for analysis and testing. I am focusing primarily on the algorithms which have been proved to have the best outcome in terms of completing the puzzles in the minimum number of moves and computing cost. With the guidelines set in terms of prioritisation I still may run into difficulties completing the algorithms but this will at least allow me to get the most important ones done first.

With the structure in place it could also allow time for the development of my own algorithms for completing the puzzles. This isn't something I am going to allocate a specific time to do as the possibilities are quite extensive. It is a component of the project in which I will consider throughout design and development as I build up the knowledge of the benefits and flaws of certain algorithms.

In the development of this project I may gain an alternative perception of prioritisation for the list of algorithms, in which will result in an adjustment of the project plan and Gantt chart. However as of now in terms of the papers that I've read I have chosen to prioritise the most efficient algorithms.

I am currently in the process of learning Prolog which is a general-purpose logic programming language associated with artificial intelligence. With a more in-depth knowledge of Prolog I may decide to switch to it from Java, this will take time to transition but may be considered fitter for purpose so will be worth doing.

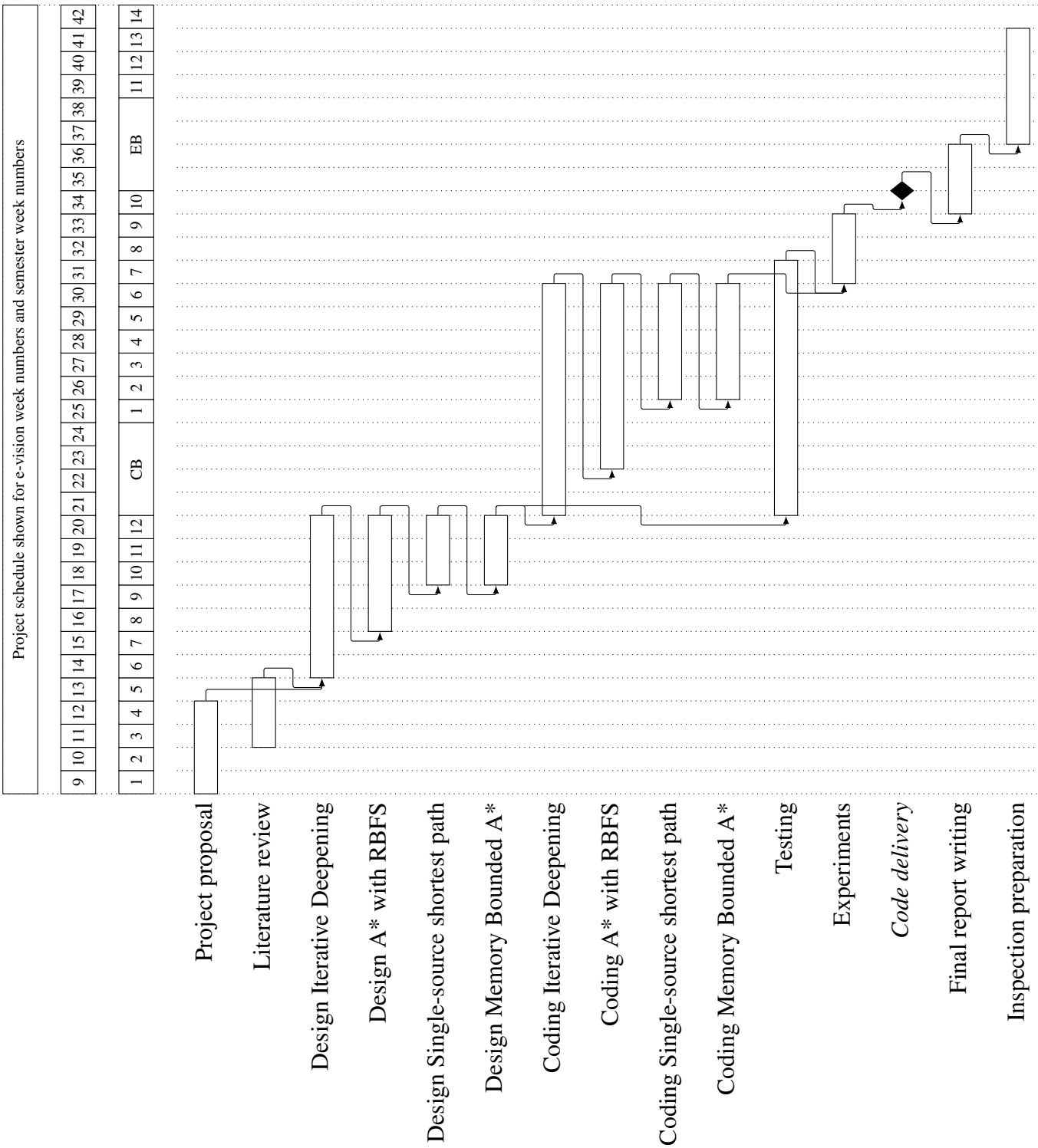

Figure 3: Project Gantt chart

References

Hatem, M., Kiesel, S., and Ruml, W. (2015). Recursive best-first search with bounded overhead. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 1151–1157.

Korf, R. E. (1993). Linear-space best-first search. *Artif. Intell.*, 62(1):41–78.

Korf, R. E. and Felner, A. (2002). Disjoint pattern database heuristics. *Artif. Intell.*, 134(1-2):9–22.

Korf, R. E., Reid, M., and Edelkamp, S. (2001). Time complexity of iterative-deepening-a*. *Artif. Intell.*, 129(1-2):199–218.

Nordhoff, B. and Lammich, P. (2012). Dijkstra's shortest path algorithm. *Archive of Formal Proofs*, 2012.

Project proposal

Description of project: aims, motivation, understanding of issues, problems	First	2.1	2.2	3	Fail
Resources, references: evidence of preliminary work to identify key resources, initial reading	First	2.1	2.2	3	Fail
Proposed approaches: relevance, suitability, appropriateness	First	2.1	2.2	3	Fail
Risks: identification, suitable contingency planning	First	2.1	2.2	3	Fail

Quality of writing

Clarity, structure correctness of writing	First	2.1	2.2	3	Fail
Presentation conforms to style	First	2.1	2.2	3	Fail

Workplan

Measurable objectives : appropriate, realistic, timely	First	2.1	2.2	3	Fail
Gantt chart: legibility, clarity, feasibility of schedule	First	2.1	2.2	3	Fail

Comments

Supervisor: supervisor

Markers should circle the appropriate level of performance in each section. Report and evaluation sheet should be collected by the student from the supervisor.