University of Southampton

COMP2208: Intelligent Systems

Comparison of Search Methods

Huw Jones

27618153

hcbj1g15@soton.ac.uk

November 30, 2016

1   Approach

In order to analyse the differences in scalability, I decided to build a framework that would allow me to minimise the time spent on writing code. At the moment, I am most familiar with Java, therefore that is the language I chose to build my solution in. My code is nowhere near "good" or optimised (in terms of real time running, not nodes expanded), but it works.

I decided to implement a monitoring thread to print out the current search status. This means that the number of nodes evaluated, how long the search has been running (in real time), and the amount of memory currently in use.

1.1   The Setup

I tried to keep data structures simple and minimalistic. The state of the puzzle is stored in a **Node**. A **Node** has a **Grid** (which represents the state of the puzzle), a parent node reference, a depth (used for IDS), and a priority (used for A*). A **Grid** has a width/height, a 2D array of characters (representing blocks), and a HashMap that maps characters to their position.

1.2   The Framework

I created a framework whereby the program parameters can be manipulated via the command line.

```
−e,  −−exit    [STATE]     Specifies exit state.
−h,  −−height  [HEIGHT]    Sets the grid height.
     −−help                Prints this help message.
−i,  −−interval [TIME]     Sets the refresh interval (in ms) − for monitoring search status.
−r,  −−ran     [STATE]     Specifies the seed used for the pseudo−random number.
−s,  −−start   [STATE]     Specifies the start state
−t,  −−type    [TYPE]      Specifies the search type:
                           BFS − Breadth First Search
                           DFS − Depth First Search
                           IDS − Iterative Deepening Search
                           A∗ − A∗ Heuristic Search
−w,  −−width   [WIDTH]     Sets the grid width.
```

This framework allowed me to create scripts to automate my searching. It also allowed me to inject different start/finish grids, as well as injecting different grid sizes - all without having to rewrite any of my code.

In addition, I allowed the input of the random number seed. This helped during debugging my program. If a DFS search didn't work with one random number, I could provide the random number and debug that case.

1.3   The Organisation

My code was organised as following. All my code is available in Appendix A.

```
|−− BlocksWorld.java
|−− blocksworld
|    |−− Block.java
|    |−− Grid.java
|    |−− GridController.java
|    |−− Node.java
|    |−− Pair.java
|    |−− Position.java
|    |−− exceptions
|    |    |−− InvalidBlockIDException.java
|    |    |−− InvalidDirectionException.java
|    |    \−− InvalidPositionException.java
|    \−− search
|         |−− AStar.java
|         |−− BFS.java
|         |−− DFS.java
|         |−− IDS.java
|         \−− Search.java
|−− run.sh
\−− test.sh
```

I chose to organise my code this way to make it easier for me to add features and group common features together.

1.4   The Scripts

I created a couple of shell scripts to aid my data collection and semi-automate running puzzles. The most useful script, `test.sh` (see Appendix A.16), took a CSV file of puzzle states and executed a given search type on the entire set, then dumped the output to a file. It saved a long time of running lots of puzzles against all the searches. `run.sh` (see Appendix A.15) is effectively a script that aliases the running of my java program.

2   Evidence

A **Grid** state is represented in a grid of dimensions $H \times W$. The agent is represented by a '`*`', blanks are represented by '`-`', and blocks are represented by a lowercase letter. In the evidence provided, the number above a **Grid** state is the node number.

When a search is running, the current status of the search is displayed. This include the number of nodes evaluated (time complexity), the length of real time the search has been running, and the amount of used memory (space complexity).

2.1   Breadth First Search

Appendix B.1 shows the order that the program evaluated nodes. It is evident that BFS is working correctly as the tiles appear to jump around if the nodes are being read in number order. Here, the first layer of the tree are nodes 1 & 2. Nodes 3 to 5 are the second layer children of node 1. Nodes 6 to 8 are the second layer children of node 2. Nodes 9 through 11 are the third layer children of node 3. And so forth for the remainder of the nodes shown.

Example output from a BFS search running is shown in Appendix C.1. Here, you can see the memory usage, my implementation of BFS fits the expected space complexity. Since the

2.2   Depth First Search

The order of nodes evaluated is shown in Appendix B.2. With these set of nodes, the movement of the agent is fluid from state to state. Therefore, it can be concluded that the implementation of DFS is working correctly.

Appendix C.2 shows the trace of a DFS running. It shows how few nodes were evaluated (in comparison to BFS), but then the solution is inherently long (compared to the optimal solution). I think this conclusively proves that my implementation of DFS is working correctly.

2.3   Iterative Deepening Search

My implementation of IDS logs when the maximum search depth increases. In the output log - as shown in Appendix B.3 - it can be seen that the nodes processed follow the expected order for IDS. When the depth is increased, it is evident that the search restarts again from the root node and proceeds to search down to the maximum depth.

In addition, the memory usage (space complexity), as shown in the output in Appendix C.3, seems to follow no trend. I believe this is due to the nodes on the fringe (at maximum depth) being removed as they aren't a solution. This would explain why the footprint of IDS stays so small when it is running.

2.4   A* Heuristic Search

With A* Heuristic Search, it is more difficult to prove that the algorithm is working correctly. Appendix B.4 shows the output log for A* Search. It is more difficult to see how my implementation of A* prioritises its node selection, however, I believe it to be working correctly. Appendix C.4 shows that very few nodes were evaluated (compared to other optimal searches such as BFS), and yet the optimal solution was found.

3   Scalability

I use the term **complexity** synonymously with difficulty in regards to searching throughout this report. In this report, the difficulty/complexity number is the length of the optimal solution for a problem.

To investigate the scalability of all 4 searches, I decided to start with the given problem and work backwards.   The base (given) problem required a minimum of 14 moves to solve and since my searches printed out the list of states to complete the puzzle, I used this printout to form puzzles of different complexities. This is how I controlled the "Problem Difficulty".

As mentioned in Section 1.4, I wrote a script to automate running puzzles of different complexities. I used the output from the base puzzle to create a CSV of states to test. These states ranged from a complexity of 1, to 14 (the base problem). I ran this test script of all 4 of the search types.

The graph in Appendix D.3 displays my findings.   It is important to note that I've used a logarithmic scale on the y axis to enhance the readability of the graph.

The most striking element of this graph is how "random" the line for DFS is. This shows that the complexity of DFS is not directly linked to the difficulty of the puzzle. This is to be expected for DFS as there is no "real" logic to how it searches – it tries random moves until it finds a solution.

The lines for BFS/IDS seem to intertwine and have the same gradient. Having the same gradient shows that both search methods increase at roughly the same exponential rate (in regards to time complexity against problem difficulty). There are times where IDS time complexity is less than BFS complexity. This is most probably due to the location of the solution node. Since BFS searches left to right, top to bottom and IDS searches from top to bottom, left to right, if the solution is on the bottom left of the tree, IDS will find it before BFS.

Finally, it's important to note how dramatic the difference between A* and BFS/IDS are in terms of time complexity. With A* the heuristics make all the difference between an efficient search and an inefficient search. In this investigation, my heuristic function is inadmissible as it is the sum of the Manhattan Distance, Depth of the Node, and the number of blocks in the incorrect position. With this heuristic function, the time complexity for A* increases exponentially with the problem difficulty, but not to the same extent as that of BFS/IDS. At a problem with difficulty of "14", A* had a time complexity 3 orders of magnitude smaller than BFS/IDS.

Overall, this graph shows the striking difference in time complexities.   I can safely conclude that the best search method, of the 4 compared, is A* Heuristic Search. It is evident that using the heuristic function mentioned previously, A* search has a time complexity magnitudes smaller, than that of IDS/BFS. This effectively rules IDS/BFS out. DFS, although sometimes providing good time complexity for some random seeds, is ultimately a poor choice if you are wanting to find the optimal solution.

4   Extras & Limitations

4.1   Extras

4.1.1   Complexity vs Problem Size

I decided to go a little further whilst investigating the complexity for different problem sizes. I collected data for puzzles on different grid sizes. I used grids sized 2x2, 3x3 and 5x5.

The graphs for these grids are available in Appendix D.1, Appendix D.2, Appendix D.3, and Appendix D.4. These graphs only reinforce my conclusions in the previous section.

With DFS, the conclusions of randomness and being erratic are shown conclusively. For example, a problem with difficulty of 1 (1 move to solve the puzzle) took DFS 2,604,225 evaluations to find a solution – the solution was 2,604,224 moves too long (compared to the optimal solution). But on the other hand, a problem that took 3.9 million evaluations using A* (as yet to be solved using DFS/IDS), solved by DFS by evaluating only 826,179 nodes. However, the major caveat being the solution from A* was 21 moves long, and the solution from DFS was 826,179 moves long.

4.1.2   Space Complexity vs Time Complexity

In this section, I decided to look at the trends between Time Complexity (number of nodes evaluated) and Space Complexity (indicated by RAM usage). The graphs are included in the Appendix. Where a trendline has been plotted, it should in theory have a y-intercept of 0, however, due to the overhead of loaded classes and the JVM, this base usage is in fact approximately 30MB.

BFS Complexity   The graph in Appendix D.5 shows a fairly linear trend. As the time complexity increases, the space complexity increases at the same rate. This is to be completely expected because a Breadth First Search does not allow for nodes to be removed from memory.

DFS Complexity   The graph in Appendix D.6 shows a linear trend. As the time complexity increases, the space complexity increases at roughly the same rate. This graph is not as conclusive due to the massive steps every so often. However, I attribute this to garbage collection.

To minimise interruptions and reduce CPU usage, garbage collection does not run all the time, but periodically. There are several measures that are watched and trigger garbage collection; these include time intervals, heap size. In the case of this graph, I believe garbage collection was being trigger when the heap grew too big. Then garbage collection kicked in and removed a lot of junk (unused/orphaned objects) from the heap.

It is obvious that my implementation of DFS is not optimal, otherwise this memory profile would be less obvious. Anyway, I believe that this graph shows that the time complexity and space complexity are linearly linked.

IDS Complexity   The graph in Appendix D.7 shows absolutely not trend whatsoever. Unlike the dodgy garbage collection in the DFS graph (Appendix D.6) I think garbage collection has worked really well for IDS.

It is noted that the maximum memory usage is 151MB. This was when IDS had evaluated just over 11 million nodes. IDS works by searching top to bottom until it reaches a limit, then left to right. If it does not find a solution, it increases its depth limit and tries again. Because of this, when IDS hits the limit on the left side of the tree, those nodes are removed from memory as it travels to the right hand side. This means that IDS has a very small space complexity footprint – and this is reflected in my findings.

A* Heuristic Complexity   The graph in **??** shows a linear trend. I attribute thickness of the "line" to be due to garbage collection – yet again.

A* works by expanding nodes, evaluating their "goodness" (also know as priority), and then expanding nodes from the "good" nodes. This strategy minimises the amount of nodes that are looked at, but means that almost all nodes end up being stored in a Priority Queue. Therefore, as the number of nodes expanded increases, then the space complexity footprint will also increase. I strongly believe that my graph shows this to be true.

4.2   Limitations

Appendices

A   Code

A.1   BlocksWorld.java

```java
import blocksworld.Grid;
import blocksworld.GridController;
import blocksworld.Pair;
import blocksworld.Position;
import blocksworld.exceptions.InvalidBlockIDException;
import blocksworld.exceptions.InvalidPositionException;
import blocksworld.search.*;
import org.omg.CORBA.DynAnyPackage.Invalid;

import java.text.ParseException;
import java.util.Arrays;
import java.util.List;

/**
 * BlocksWorld
 *
 * @author Huw Jones
 * @since 08/10/2016
 */
public class BlocksWorld {

    public static void main(String[] args) {
        List<String> argList = Arrays.asList(args);

        BlocksWorld.header();

        try {
            if (argList.contains("--help")) {
                help();
                return;
            }

            int width = -1;
            int height = -1;
            int refresh = -1;

            Long seed = null;
            if (argList.contains("--height") || argList.contains("-h")) {
                int index = (argList.contains("-h")) ? argList.indexOf("-h") : argList.indexOf
                    ("--height");
                height = getInt(argList, index);
            }
            if (argList.contains("--width") || argList.contains("-w")) {
                int index = (argList.contains("-w")) ? argList.indexOf("-w") : argList.indexOf
                    ("--width");
                width = getInt(argList, index);
            }

            if (argList.contains("--interval") || argList.contains("-i")) {
                int index = (argList.contains("-i")) ? argList.indexOf("-i") : argList.indexOf
                    ("--interval");
                refresh = getInt(argList, index);
            }

            if (width == -1 || height == -1) {
                System.out.println("Please specify width/height.");
                return;
            }

            if (argList.contains("--ran") || argList.contains("-r")) {
                int index = (argList.contains("-r")) ? argList.indexOf("-r") : argList.indexOf
                    ("--ran");
                seed = getSeed(argList, index);
            }

            if (!argList.contains("--type") && !argList.contains("-t")) {
                System.out.println("Please specify a search type. (See help for more details)"
                    );
```

```java
                    return;
            }

            int index = (argList.contains("-t")) ? argList.indexOf("-t") : argList.indexOf("--
                type");
            try {

                String type = argList.get(index + 1);
                Grid startGrid = null;
                Grid exitGrid = null;
                if (argList.contains("--start") || argList.contains("-s")){
                    int startIndex = (argList.contains("-s")) ? argList.indexOf("-s") :
                        argList.indexOf("--start");
                    startGrid = parseState(argList.get(startIndex + 1), width, height);
                }
                if(argList.contains("--exit") || argList.contains("-e")) {
                    int exitIndex = (argList.contains("-e")) ? argList.indexOf("-e") : argList
                        .indexOf("--exit");
                    exitGrid = parseState(argList.get(exitIndex + 1), width, height);
                }

                search(type, startGrid, exitGrid, seed, refresh);
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("No option was specified for " + argList.get(index));
            }
        } catch (ParseException ex) {
            System.out.println("Failed to read input: " + ex.getMessage());
        } catch (ArrayIndexOutOfBoundsException ex) {
            System.out.println("No argument specified: " + ex.getMessage());
        } catch (IllegalArgumentException ex) {
            System.out.println("No input provided for option: " + ex.getMessage());
        }
    }

    /**
     * Prints out programme header info
     */
    private static void header() {
        System.out.println("Usage: BlocksWorld [OPTION]...");
        System.out.println("COMP2208 BlocksWorld Search Tool.\n");
    }

    /**
     * Prints out help
     */
    private static void help() {
        BlocksWorld.header();
        System.out.println("Arguments:");
        System.out.println("  -e, --exit [STATE]\tSpecifies exit state.");
        System.out.println("  -h, --height [HEIGHT]\tSets the grid height.");
        System.out.println("      --help\t\tPrints this help message.");
        System.out.println("  -i, --interval [TIME]\tSets the refresh interval (in ms) for
            monitoring search status.");
        System.out.println("  -r, --ran [STATE]\tSpecifies the seed used for the pseudo-random
             number.");
        System.out.println("  -s, --start [STATE]\tSpecifies the start state");
        System.out.println("  -t, --type [TYPE]\tSpecifies the search type:\r\n\t\t\tBFS -
            Breadth First Search\r\n\t\t\tDFS - Depth First Search\r\n\t\t\tIDS - Iterative
            Deepening Search\r\n\t\t\tA* - A* Heuristic Search");
        System.out.println("  -w, --width [WIDTH]\tSets the grid width.");
    }

    private static int getInt(List<String> args, int argIndex) throws ParseException,
        ArrayIndexOutOfBoundsException, IllegalArgumentException {
        String widthStr = args.get(argIndex + 1);
        if (widthStr.substring(0, 1).equals("'")) {
            throw new IllegalArgumentException("No option was specified for " + args.get(
                argIndex));
        }
        return Integer.parseInt(widthStr);
    }

    private static long getSeed(List<String> args, int argIndex) throws ParseException,
        ArrayIndexOutOfBoundsException, IllegalArgumentException {
        String seedStr = args.get(argIndex + 1);
        return Long.parseLong(seedStr);
```

```java
    }

    private static Grid parseState(String state, int src_width, int src_height) throws
        ParseException {
        List<String> substrs = Arrays.asList(state.split(":"));
        try {
            int width = Integer.parseInt(substrs.get(0));
            int height = Integer.parseInt(substrs.get(1));
            if(width != src_width || height != src_height){
                throw new ParseException("Height/Width in state does not match provided height
                    /width.", 0);
            }
            Grid g = GridController.createGrid(width, height);

            String row;
            char symbol;
            for (int i = 2; i < substrs.size(); i++) {
                row = substrs.get(i);
                for (int x = 0; x < g.getWidth(); x++) {
                    symbol = row.charAt(x);
                    try {
                        if (symbol >= 'a' && symbol <= 'z') {
                            g.placeBlock(symbol, new Position(x, i - 2));
                        } else if (symbol == '*') {
                            try {
                                g.placeAgent(x, i - 2);
                            } catch (InvalidBlockIDException ex){
                                System.out.println("Multiple agents detected ... skipping");
                            }
                        }
                    } catch (InvalidPositionException e) {
                        e.printStackTrace();
                    }
                }
            }
            return g;
        } catch (NumberFormatException ex) {
            throw new ParseException(ex.getMessage(), 0);
        }
    }

    private static void search(String type, Grid startState, Grid exitState, Long seed, int
        refreshTime) {
        Search search = null;
        switch (type) {
            case "BFS":
                search = new BFS();
                break;
            case "DFS":
                search = new DFS();
                break;
            case "IDS":
                search = new IDS();
                break;
            case "A*":
                search = new AStar();
                break;
            default:
                header();
                System.out.println(String.format("Type '%s' was not recognised.", type));
                return;
        }
        if (search == null) return;
        if (startState != null) {
            search.setStartState(startState);
        }
        if (exitState != null) {
            search.setExitState(exitState);
        }
        if(seed != null){
            search.setSeed(seed);
        }
        if(refreshTime != -1L){
            search.setRefreshTime(refreshTime);
        }
        search.run();
```

```
    }
}
```

A.2   Grid.java

```java
package blocksworld;

import blocksworld.exceptions.InvalidBlockIDException;
import blocksworld.exceptions.InvalidPositionException;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.NoSuchElementException;
import java.util.stream.Collectors;

/**
 * Grid
 * Holds the state of the grid
 *
 * @author Huw Jones
 * @since 08/10/2016
 */
public class Grid {

    private int width;
    private int height;
    private char[][] grid;
    private HashMap<Character, Position> blocks;

    /**
     * Creates a new grid with a specified width and height
     *
     * @param width  Width of new grid
     * @param height Height of new grid
     */
    public Grid(int width, int height) {
        this.width = width;
        this.height = height;
        this.grid = new char[width][height];
        this.blocks = new HashMap<>();
    }

    /**
     * Returns the width of the grid
     *
     * @return Grid Width
     */
    public int getWidth() {
        return width;
    }

    /**
     * Returns the height of the grid
     *
     * @return Grid Height
     */
    public int getHeight() {
        return height;
    }

    /**
     * Places the agent to the grid at position (x, y)
     *
     * @param x x-coord
     * @param y y-coord
     * @throws InvalidPositionException Thrown if the given position is invalid
     */
    public void placeAgent(int x, int y) throws InvalidPositionException,
        InvalidBlockIDException {
        placeAgent(new Position(x, y));
    }

    /**
     * Places the agent to the grid at position P(x, y)
     *
     * @param position Position P(x, y)
     * @throws InvalidPositionException Thrown if the given position is invalid
     */
    public void placeAgent(Position position) throws InvalidPositionException,
```

```java
        InvalidBlockIDException {
        if(this.blocks.containsKey('*')){
            throw new InvalidBlockIDException('*');
        }
        this.placeBlock('*', position);
    }

    /**
     * Adds a block to the grid at position (x, y) with the ID blockID.
     * Throws an exception if the position is invalid
     *
     * @param blockID  ID of the block, must be unique
     * @param position Position of the block
     * @throws InvalidPositionException Thrown if the position is invalid
     */
    public void placeBlock(char blockID, Position position) throws InvalidPositionException {
        if (!isPositionValid(position)) {
            throw new InvalidPositionException(position);
        }
        try {
            this.grid[position.getX()][position.getY()] = blockID;
        } catch (ArrayIndexOutOfBoundsException ex) {
            ex.printStackTrace();
        }

        // Prevent adding null chars to the block HashMap
        if (blockID == Character.MIN_VALUE) return;
        this.blocks.put(blockID, position);
    }

    /**
     * Returns whether a position is valid on the grid
     *
     * @param position Position to check
     * @return False if position is not valid
     */
    public boolean isPositionValid(Position position) {
        return isPositionValid(position.getX(), position.getY());
    }

    /**
     * Returns whether a position is valid on the grid
     *
     * @param x x-coord
     * @param y y-coord
     * @return False if position is not valid
     */
    public boolean isPositionValid(int x, int y) {
        return (x < this.width && y < this.height && x >= 0 && y >= 0);
    }

    /**
     * Places a block on the grid
     *
     * @param block Block to place
     * @throws InvalidPositionException Thrown if the position is invalid
     */
    public void placeBlock(Block block) throws InvalidPositionException {
        placeBlock(block.getID(), block.getPosition());
    }

    /**
     * Gets the agent block
     *
     * @return Agent block
     * @throws NoSuchElementException If the block was not found
     */
    public Block getAgent() throws NoSuchElementException {
        return getBlock('*');
    }

    /**
     * Gets the Block with blockID
     *
     * @param blockID Block to fetch
     * @return Block
```

```java
     * @throws NoSuchElementException If block was not found
     */
    public Block getBlock(char blockID) throws NoSuchElementException {
        if (!this.blocks.containsKey(blockID)) {
            throw new NoSuchElementException("No_such_block_with_ID:_" + blockID);
        }
        return new Block(blockID, this.blocks.get(blockID));
    }

    /**
     * Gets the list of blocks in the grid
     *
     * @return List of Blocks
     */
    public ArrayList<Block> getBlocks() {
        // Map the HashMap to an ArrayList<Block>
        return this.blocks.entrySet().stream().map(map -> new Block(map.getKey(), map.getValue
            ())).collect(Collectors.toCollection(ArrayList::new));
    }

    @Override
    public String toString() {
        String grid = "";
        try {
            Character block;
            for (int y = 0; y < this.height; y++) {
                for (int x = 0; x < this.width; x++) {
                    block = this.grid[x][y];
                    grid += (block != Character.MIN_VALUE) ? block : "-";

                }
                grid += "\n";
            }
        } catch (Exception ex){
            ex.printStackTrace();
        }
        return grid;
    }
}
```

A.3   GridController.java

```java
package blocksworld;

import blocksworld.exceptions.InvalidBlockIDException;
import blocksworld.exceptions.InvalidDirectionException;
import blocksworld.exceptions.InvalidPositionException;

import java.util.ArrayList;

/**
 * Grid Controller
 *
 * @author Huw Jones
 * @since 08/10/2016
 */
public class GridController {

    public static Grid placeBlock(Grid grid, char blockID, Position position) throws
        InvalidPositionException {
        grid.placeBlock(blockID, position);
        return grid;
    }

    public static Grid placeBlock(Grid grid, char blockID, int x, int y) throws
        InvalidPositionException {
        grid.placeBlock(blockID, new Position(x, y));
        return grid;
    }

    public static Grid placeBlock(Grid grid, Block block) throws InvalidPositionException {
        grid.placeBlock(block.getID(), block.getPosition());
        return grid;
    }

    public static Grid placeAgent(Grid grid, int x, int y) throws InvalidPositionException,
        InvalidBlockIDException {
        grid.placeAgent(x, y);
        return grid;
    }

    public static Grid placeAgent(Grid grid, Position position) throws
        InvalidPositionException, InvalidBlockIDException {
        grid.placeAgent(position);
        return grid;
    }

    public static Grid move(Grid grid, DIRECTION direction) throws InvalidDirectionException {
        if (!canMove(grid, direction))
            throw new InvalidDirectionException(direction, grid.getAgent().getPosition(),
                getNewAgentPosition(grid, direction));

        // Get position where agent is *going* to move to
        Position newAgentPosition = getNewAgentPosition(grid, direction);

        Grid newGrid = GridController.createGrid(grid.getWidth(), grid.getHeight());

        ArrayList<Block> oldBlocks = grid.getBlocks();
        ArrayList<Block> newBlocks = new ArrayList<>();

        Block oldBlock = null;
        Block oldAgent = null;

        for (Block block : oldBlocks) {
            if (block.getID() == '*') {
                oldAgent = block;
            } else if (block.getPosition().equals(newAgentPosition)) {
                oldBlock = block;
            } else {
                newBlocks.add(block);
            }
        }
        if (oldAgent != null) {
            newBlocks.add(new Block('*', newAgentPosition));
            if (oldBlock != null) {
                newBlocks.add(new Block(oldBlock.getID(), oldAgent.getPosition()));
```

```java
            }
        }

        for (Block block : newBlocks) {
            try {
                newGrid.placeBlock(block);
            } catch (InvalidPositionException e) {
                e.printStackTrace();
            }
        }

        return newGrid;
    }

    public static boolean canMove(Grid grid, DIRECTION direction) {
        try {
            getNewAgentPosition(grid, direction);
        } catch (InvalidDirectionException ex) {
            return false;
        }
        return true;
    }

    private static Position getNewAgentPosition(Grid grid, DIRECTION direction) throws
        InvalidDirectionException {
        Position oldPosition = grid.getAgent().getPosition();
        int old_x = oldPosition.getX();
        int old_y = oldPosition.getY();

        int new_x = -1;
        int new_y = -1;

        switch (direction) {
            case NORTH:
                new_x = old_x;
                new_y = old_y - 1;
                break;
            case EAST:
                new_x = old_x + 1;
                new_y = old_y;
                break;
            case SOUTH:
                new_x = old_x;
                new_y = old_y + 1;
                break;
            case WEST:
                new_x = old_x - 1;
                new_y = old_y;
                break;
        }

        Position newPosition = new Position(new_x, new_y);

        if (!grid.isPositionValid(new_x, new_y))
            throw new InvalidDirectionException(direction, oldPosition, newPosition);

        return newPosition;
    }

    public static Grid createGrid(int width, int height) {
        return new Grid(width, height);
    }

    public enum DIRECTION {
        NORTH,
        EAST,
        SOUTH,
        WEST
    }
}
```

A.4 Node.java

```java
package blocksworld;

/**
 * Node
 * Stores a grid state
 *
 * @author Huw Jones
 * @since 08/10/2016
 */
public class Node {

    private Grid grid = null;
    private Node parent;
    private int depth;
    private Integer priority = null;

    private Node() {
        this.depth = 0;
    }

    public Node(Node parent) {
        this.parent = parent;
        // Cheap way to calculate depth of this node
        this.depth = parent.getDepth() + 1;
    }

    public int getDepth() {
        return depth;
    }

    public static Node createRootNode() {
        return new Node();
    }

    public Grid getGrid() {
        return grid;
    }

    public void setGrid(Grid grid) {
        if (grid != null) this.grid = grid;
    }

    public int getPriority() {
        return priority;
    }

    public void setPriority(int priority) {
        if (this.priority == null) {
            this.priority = priority;
        }
    }

    public Node getParent() {
        return parent;
    }
}
```

A.5   Pair.java

```java
package blocksworld;

/**
 * Holds a Pair of Values
 *
 * @author Huw Jones
 * @since 08/10/2016
 */
public class Pair<K, V> {

    private K key;
    private V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() {
        return key;
    }

    public V getValue() {
        return value;
    }

    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof Pair)) return false;
        Pair p = (Pair) obj;
        return this.key == p.key && this.value == p.value;
    }

    @Override
    public String toString() {
        return String.format("(%s, %s)", key, value);
    }
}
```

A.6   Position.java

```java
package blocksworld;

/**
 * Effectively a Co-Ordinate in 2D space
 *
 * @author Huw Jones
 * @since 04/11/2016
 */
public class Position extends Pair<Integer, Integer> {
    public Position(int x, int y) {
        super(x, y);
    }

    public int getX(){
        return this.getKey();
    }

    public int getY(){
        return this.getValue();
    }

    public Position subtract(Position position){
        return new Position(
                this.getX() - position.getX(),
                this.getY() - position.getY()
        );
    }
}
```

A.7 InvalidBlockIDException.java

```java
package blocksworld.exceptions;

/**
 * Thrown if an invalid Block ID was specified
 *
 * @author Huw Jones
 * @since 08/10/2016
 */
public class InvalidBlockIDException extends Exception {

    /**
     * Constructs a new exception with {@code null} as its detail message.
     * The cause is not initialized, and may subsequently be initialized by a
     * call to {@link #initCause}.
     */
    public InvalidBlockIDException(char c) {
        super("Invalid_BlockID:_" + c);
    }
}
```

A.8 InvalidDirectionException.java

```java
package blocksworld.exceptions;

import blocksworld.GridController;
import blocksworld.Pair;

/**
 * Thrown if an invalid direction was given
 *
 * @author Huw Jones
 * @since 08/10/2016
 */
public class InvalidDirectionException extends Exception {

    public InvalidDirectionException(GridController.DIRECTION d, Pair oldPos, Pair newPos) {
        super("Cannot_move_in_direction:_" + d + ",_from_position_" + oldPos + ",_to_" +
            newPos);
    }
}
```

A.9 InvalidPositionException.java

```java
package blocksworld.exceptions;

import blocksworld.Position;

/**
 * Thrown if an invalid position was specified
 *
 * @author Huw Jones
 * @since 08/10/2016
 */
public class InvalidPositionException extends Exception {

    public InvalidPositionException(int x, int y) {
        super(String.format("Invalid_position_at_(%d,_%d)", x, y));
    }

    public InvalidPositionException(Position position) {
        this(position.getX(), position.getY());
    }
}
```

A.10   AStar.java

```java
package blocksworld.search;

import blocksworld.*;
import blocksworld.exceptions.InvalidDirectionException;

import java.util.*;

/**
 * A* Search
 *
 * @author Huw Jones
 * @since 27/11/2016
 */
public class AStar extends Search {

    PriorityQueue<Node> nodeQueue;

    /**
     * Set up the initial environment before running the search
     */
    @Override
    protected void preRun() {
        this.nodeQueue = new PriorityQueue<>(new PriorityComparator());
        this.rootNode = Node.createRootNode();
        this.rootNode.setGrid(this.startGrid);
    }

    /**
     * Where the actual search runs
     */
    @Override
    protected void runSearch() throws Exception {
        ArrayList<GridController.DIRECTION> directions = new ArrayList<>(4);
        Arrays.stream(GridController.DIRECTION.values()).forEach(directions::add);

        this.currentNode = rootNode;
        while_loop:
        while (true) {
            numberOfNodes++;

            // Check the if the node satisfies the exit condition
            if (this.checkExitCondition(currentNode.getGrid())) {
                completed(currentNode);
                break;
            }

            for (GridController.DIRECTION direction : directions) {
                try {
                    // Process the move and store the new state in the node
                    Node newNode = new Node(currentNode);
                    newNode.setGrid(
                            GridController.move(
                                    newNode.getParent().getGrid(),
                                    direction
                            )
                    );

                    // Calculate the node heuristic score and add it to the queue
                    newNode.setPriority(
                            calculatePriority(newNode)
                    );
                    nodeQueue.add(newNode);
                } catch (InvalidDirectionException e) {
                }
            }
            // Process the next node
            nextNode();
        }
    }

    @Override
    protected void nextNode() {
        currentNode = nodeQueue.poll();
    }
```

```java
    /**
     * Calculates the priority (heuristic score) of the node
     *
     * @param node Node to calculate score for
     * @return Score for that node
     */
    private int calculatePriority(Node node) {
        int score = 0;
        score += getManhattanDistance(node.getGrid());
        score += getTilesInCorrectPlace(node.getGrid());
        score += node.getDepth();
        return score;
    }

    /**
     * Calculates the Manhattan Distance Heuristic
     *
     * @param grid Grid to calculate
     * @return score
     */
    private int getManhattanDistance(Grid grid) {
        int score = 0;
        ArrayList<Block> blocks = grid.getBlocks();
        for (Block block : blocks) {
            try {
                // Get the difference between the exit position and the current block position
                Position difference = this.exitGrid
                        .getBlock(block.getID())
                        .getPosition()
                        .subtract(block.getPosition());
                // Add the X/Y distance from target block position (distance not displacement,
                        hence Math.abs)
                score += Math.abs(difference.getX());
                score += Math.abs(difference.getY());
            } catch (NoSuchElementException ex) {
            }
        }
        return score;
    }

    /**
     * Calculates the number of tiles in the correct place
     *
     * @param grid Grid to calculate
     * @return score
     */
    private int getTilesInCorrectPlace(Grid grid) {
        ArrayList<Block> blocks = grid.getBlocks();
        int score = 0;
        for (Block block : blocks) {
            try {
                // Increment score for every incorrectly positioned block
                if (!this.exitGrid.getBlock(block.getID()).getPosition().equals(block.
                        getPosition())) score++;
            } catch (NoSuchElementException ex) {

            }
        }
        return score;
    }

    /**
     * Finds the highest priority node (node with lowest score)
     * Used to sort the PriorityQueue
     */
    private class PriorityComparator implements Comparator<Node> {
        @Override
        public int compare(Node o1, Node o2) {
            return o1.getPriority() - o2.getPriority();
        }
    }
}
```

A.11   BFS.java

```java
package blocksworld.search;

import blocksworld.GridController;
import blocksworld.GridController.DIRECTION;
import blocksworld.Node;
import blocksworld.Pair;
import blocksworld.exceptions.InvalidDirectionException;

import java.util.Queue;
import java.util.concurrent.ConcurrentLinkedQueue;

/**
 * Breadth First Search
 *
 * @author Huw Jones
 * @since 21/10/2016
 */
public class BFS extends Search {

    private Queue<Pair<Node, DIRECTION>> nodeQueue;

    /**
     * Set up the initial environment before running the search
     */
    @Override
    protected void preRun() {
        this.nodeQueue = new ConcurrentLinkedQueue<>();
        this.rootNode = Node.createRootNode();
        this.rootNode.setGrid(this.startGrid);
    }

    /**
     * Where the actual search runs
     */
    @Override
    protected void runSearch() {
        this.currentNode = rootNode;

        while (true) {
            if (currentDirection != null) {
                try {
                    currentNode.setGrid(
                            GridController.move(
                                    currentNode.getParent().getGrid(),
                                    currentDirection
                            )
                    );
                    numberOfNodes++;
                    if (this.checkExitCondition(currentNode.getGrid())) {
                        completed(currentNode);
                        break;
                    }
                } catch (InvalidDirectionException e) {
                    nextNode();
                    continue;
                }
            }

            for (DIRECTION direction : DIRECTION.values()) {
                nodeQueue.add(new Pair<>(new Node(currentNode), direction));
            }
            nextNode();
        }
    }

    @Override
    protected void nextNode() {
        currentPair = nodeQueue.poll();
        currentNode = currentPair.getKey();
        currentDirection = currentPair.getValue();
    }
}
```

A.12   DFS.java

```java
package blocksworld.search;

import blocksworld.GridController;
import blocksworld.GridController.DIRECTION;
import blocksworld.Node;
import blocksworld.Pair;
import blocksworld.exceptions.InvalidDirectionException;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Stack;

/**
 * Depth First Search
 *
 * @author Huw Jones
 * @since 27/10/2016
 */
public class DFS extends Search {

    private Stack<Pair<Node, DIRECTION>> nodeStack;

    @Override
    protected void preRun() {
        this.nodeStack = new Stack<>();
        this.rootNode = Node.createRootNode();
        this.rootNode.setGrid(this.startGrid);
    }

    @Override
    protected void runSearch() {
        // Init
        ArrayList<DIRECTION> directions = new ArrayList<>(4);
        Arrays.stream(DIRECTION.values()).forEach(directions::add);

        currentNode = rootNode;
        while (true) {


            if (currentDirection != null) {
                try {
                    currentNode.setGrid(
                            GridController.move(
                                    currentNode.getParent().getGrid(),
                                    currentDirection
                            )
                    );
                    numberOfNodes++;
                    if (this.checkExitCondition(currentNode.getGrid())) {
                        completed(currentNode);
                        break;
                    }
                } catch (InvalidDirectionException e) {
                    nextNode();
                    continue;
                }
            }

            Collections.shuffle(directions, this.random);

            for (DIRECTION direction : directions) {
                nodeStack.push(new Pair<>(new Node(currentNode), direction));
            }

            nextNode();
        }
    }

    @Override
    protected void nextNode() {
        currentPair = nodeStack.pop();
        currentNode = currentPair.getKey();
```

```
            currentDirection = currentPair.getValue();
    }
}
```

A.13   IDS.java

```java
package blocksworld.search;

import blocksworld.GridController;
import blocksworld.GridController.DIRECTION;
import blocksworld.Node;
import blocksworld.Pair;
import blocksworld.exceptions.InvalidDirectionException;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Stack;

/**
 * Iterative Deepening Search
 *
 * @author Huw Jones
 * @since 12/11/2016
 */
public class IDS extends Search {

    private Stack<Pair<Node, DIRECTION>> nodeStack;
    private ArrayList<DIRECTION> directions;
    private int depth;

    /**
     * Set up the initial environment before running the search
     */
    @Override
    protected void preRun() {
        this.nodeStack = new Stack<>();
        this.rootNode = Node.createRootNode();
        this.rootNode.setGrid(this.startGrid);
        directions = new ArrayList<>(4);
        Arrays.stream(DIRECTION.values()).forEach(directions::add);

        currentNode = rootNode;
        currentPair = null;
        currentDirection = null;
    }

    /**
     * Where the actual search runs
     */
    @Override
    protected void runSearch() {
        while (true) {
            // Check if we've hit the depth limit
            if (currentNode.getDepth() > depth) {
                // If we have no more nodes in the stack, increase the depth
                if(this.nodeStack.size() == 0) {
                    increaseDepth();
                } else {
                    // Otherwise continue processing the stack
                    nextNode();
                }
                continue;
            }


            if (currentDirection != null) {
                try {
                    // Process the move and store the new state in the node
                    currentNode.setGrid(
                            GridController.move(
                                    currentNode.getParent().getGrid(),
                                    currentDirection
                            )
                    );
                    numberOfNodes++;
                    // Check if the grid meets the exit condition, if so, exit the search
                    if (this.checkExitCondition(currentNode.getGrid())) {
                        completed(currentNode);
```

```java
                        break;
                    }
                } catch (InvalidDirectionException e) {
                    if (nodeStack.size() == 0){
                        increaseDepth();
                    } else {
                        nextNode();
                    }
                    continue;
                }
            }

            Collections.shuffle(directions, this.random);

            // Push new directions on the stack to be processed
            for (DIRECTION direction : directions) {
                nodeStack.push(new Pair<>(new Node(currentNode), direction));
            }

            nextNode();
        }
        System.out.println("Max_Iterative_Depth:");
        System.out.println(depth);
    }

    /**
     * Gets the next node off of the stack
     */
    @Override
    protected void nextNode() {
        currentPair = nodeStack.pop();
        currentNode = currentPair.getKey();
        currentDirection = currentPair.getValue();
    }

    /**
     * Increases the depth of the search
     */
    private void increaseDepth(){
        // Reset the search environment
        preRun();
        // Increment depth
        depth++;
        // Log new depth
        System.out.println("\r\nDepth_increased:_"+ depth);
    }
}
```

A.14   Search.java

```java
package blocksworld.search;

import blocksworld.*;
import blocksworld.exceptions.InvalidBlockIDException;
import blocksworld.exceptions.InvalidPositionException;

import java.text.NumberFormat;
import java.util.List;
import java.util.Locale;
import java.util.Random;
import java.util.Stack;
import java.util.stream.Collectors;

/**
 * Abstract Search Class
 *
 * @author Huw Jones
 * @since 11/10/2016
 */
public abstract class Search {

    protected Grid startGrid;
    protected long randomSeed;
    protected Random random;
    protected long numberOfNodes = 0;
    protected Grid exitGrid;
    protected Node rootNode;
    protected Node currentNode;
    protected Pair<Node, GridController.DIRECTION> currentPair;
    protected GridController.DIRECTION currentDirection = null;
    private boolean completed = false;
    private long startTime;
    private int refreshTime = 100;

    public Search() {
        this.randomSeed = new Random().nextLong();
        this.buildGrid();
        this.createExitGrid();
        Runtime.getRuntime().addShutdownHook(new Thread() {
            @Override
            public void run() {
                System.out.print("\r\n\r\n");
            }
        });
    }

    /**
     * Builds the default 4x4 grid
     */
    private void buildGrid() {
        startGrid = GridController.createGrid(4, 4);
        try {
            GridController.placeBlock(startGrid, 'a', 0, 3);
            GridController.placeBlock(startGrid, 'b', 1, 3);
            GridController.placeBlock(startGrid, 'c', 2, 3);
            GridController.placeAgent(startGrid, 3, 3);
        } catch (InvalidPositionException | InvalidBlockIDException e) {
            e.printStackTrace();
        }
    }

    /**
     * Builds the defualt exit grid state
     */
    void createExitGrid() {
        this.exitGrid = GridController.createGrid(this.startGrid.getWidth(), this.startGrid.
            getHeight());

        try {
            GridController.placeBlock(exitGrid, 'a', 1, 1);
            GridController.placeBlock(exitGrid, 'b', 1, 2);
            GridController.placeBlock(exitGrid, 'c', 1, 3);
        } catch (InvalidPositionException e) {
            e.printStackTrace();
```

```java
        }
    }

    /**
     * Sets the random number seed
     * @param seed Seed
     */
    public void setSeed(long seed) {
        this.randomSeed = seed;
    }

    /**
     * Runs the search
     */
    public void run() {
        System.out.println("Creating random seed ...");
        this.random = new Random(this.randomSeed);
        System.out.println(String.format("Random seed: %d.", this.randomSeed));
        System.out.println("Running Search::preRun");
        this.preRun();
        System.out.println("Start State:");
        System.out.println(this.startGrid.toString());
        System.out.println("Exit State:");
        System.out.println(this.exitGrid.toString());
        System.out.println("Running Search::runSearch");
        try {
            Thread t = new Thread(new Monitor(), "MonitorThread");
            t.setDaemon(true);
            this.startTime = System.nanoTime();
            t.start();
            this.runSearch();
        } catch (Exception ex) {
            System.out.println("Error running search.");
            ex.printStackTrace();
        }
    }

    /**
     * Called when a search completes.
     * It dumps the solution and stats to console
     * @param exitNode Node that solves the puzzle
     */
    protected void completed(Node exitNode) {
        this.completed = true;
        System.out.println("\r\n\r\n========================================");
        System.out.println("Solution found.");

        System.out.println("Solution as follows:");
        System.out.println(this.getSolution(exitNode));

        System.out.println("\r\n========================================");
        System.out.println("Start State:\r\n");
        System.out.println(this.startGrid.toString());
        System.out.println("========================================");
        System.out.println("Exit State:\r\n");
        System.out.println(this.exitGrid.toString());
        System.out.println("========================================");
        System.out.println("Random Seed:");
        System.out.println(this.randomSeed);
        System.out.println("========================================");
        System.out.println("Nodes Expanded:");
        System.out.println(this.numberOfNodes);
        System.out.println("========================================");
    }

    /**
     * Set up the initial environment before running the search
     */
    abstract protected void preRun();

    /**
     * Where the actual search runs
     */
    abstract protected void runSearch() throws Exception;

    /**
```

```
 * Builds the solution from the exit node
 * @param endNode End Node that is in the exit state
 * @return String that is the solution
 */
public String getSolution(Node endNode) {
    // Using a stack so we can reverse the order of the nodes easier
    Stack<String> states = new Stack<>();
    Node currentNode = endNode;

    // Dump all nodes on the stack whilst the parent isn't null
    // Only the root node has a null parent
    do {
        if (currentNode != null) {
            if (currentNode.getGrid() != null) {
                states.add(currentNode.getGrid().toString());
            }
        }
    } while ((currentNode = currentNode.getParent()) != null);

    // Count moves whilst looping through the stack and append grid state to the sting
    StringBuilder builder = new StringBuilder();
    String currentString;
    int moves = 0;
    while (states.size() != 0) {
        currentString = states.pop();
        builder.append("\n");
        builder.append(moves);
        builder.append(":");
        builder.append("\n");
        builder.append(currentString);
        moves++;
    }
    return builder.toString();
}

/**
 * Checks whether or not a grid meets the exit criteria
 * @param grid Grid to check
 * @return true if the exit condition has been reached
 */
protected boolean checkExitCondition(Grid grid) {
    // Lambda to get blocks (excluding the agent "*")
    List<Block> blocks = grid.getBlocks().stream().filter(b -> b.getID() != '*').collect(
        Collectors.toList());

    // Assume we're complete
    boolean exitReached = true;
    Block comparisonBlock;

    // Loop through the block and AND the matching block result
    for (Block block : blocks) {
        comparisonBlock = this.exitGrid.getBlock(block.getID());
        exitReached &= comparisonBlock.getPosition().equals(block.getPosition());
    }

    return exitReached;
}

/**
 * Gets the monitor thread interval refresh time
 * @return Interval refresh time (in ms)
 */
public int getRefreshTime() {
    return refreshTime;
}

/**
 * Sets the refresh interval
 * @param time time in ms
 */
public void setRefreshTime(int time) {
    this.refreshTime = time;
}

/**
 * Sets the start grid state
```

```java
     * @param startGrid  Grid  to  start  from
     */
    public  void  setStartState(Grid  startGrid)  {
        this.startGrid  =  startGrid;
    }

    /**
     * Sets  the  exit  grid  state
     * @param exitGrid  Grid  that  forms  the  exit  conditions
     */
    public  void  setExitState(Grid  exitGrid)  {
        this.exitGrid  =  exitGrid;
    }

    /**
     * Gets  the  next  node
     */
    protected  abstract  void  nextNode();

    /**
     * Monitor  Thread  (provides  ongoing  stats  of  the  search  in  console)
     */
    private  class  Monitor  implements  Runnable  {
        @Override
        public  void  run()  {
            long  time;
            long  minutes;
            long  seconds;
            long  memory;
            while  (!completed)  {
                time  =  System.nanoTime()  −  startTime;
                memory  =  (Runtime.getRuntime().totalMemory()  −  Runtime.getRuntime().freeMemory
                    ())  /  1048576;
                seconds  =  time  /  1000000000;
                minutes  =  seconds  /  60;
                seconds  −=  minutes  ∗  60;
                System.out.print(String.format("\rExpanded␣Nodes:␣%12s\t\tElapsed␣Time␣[%s:%s
                    ]\t\tUsed␣Memory:␣%6sMB",
                        NumberFormat.getNumberInstance(Locale.getDefault()).format(
                            numberOfNodes),
                        String.format("%2d",  minutes).replace('␣',  '0'),
                        String.format("%2d",  seconds).replace('␣',  '0'),
                        NumberFormat.getNumberInstance(Locale.getDefault()).format(memory)));
                try  {
                    Thread.sleep(Search.this.getRefreshTime());
                }  catch  (InterruptedException  e)  {
                }
            }
        }
    }
}
```

A.15   run.sh

```bash
#!/bin/bash
java -cp "out/production/COMP2208" -XX:+UseG1GC BlocksWorld $*
```

A.16   test.sh

```bash
#!/bin/bash
fFlag=false;
tFlag=false;
nFlag=false;
output=".";
testNumber="";
interval=100;

while getopts 'i:n:t:f:o:' flag; do
  case "${flag}" in
    i) interval="${OPTARG}" ;;
    o) output="${OPTARG}" ;;
    t) type="${OPTARG}" ; tFlag=true ;;
    f) file="${OPTARG}" ; fFlag=true ;;
    n) testNumber="${OPTARG}" ; nFlag=true ;;
    *) echo "Unexpected option ${flag}" ; exit ;
  esac
done

if ! $fFlag
then
  echo "File of states must be provided with -f!";
  exit 1
fi

if ! $tFlag
then
  echo "Search type must be provided with -t!";
  exit 1
fi

if ! $nFlag
then
  echo "Test number must be provided with -n!";
  exit 1
fi

cat "$file" | awk '
BEGIN{
  RS = "\n";
  FS = ",";
}
{
  output="'$output'/'$type'-"$1".txt";
  split($2, parts, ":");
  height=parts[2];
  width=parts[1];
  if($1 == "" || $1 == "'$testNumber'") {
    if(NF == 2){
      system("./run.sh -i '$interval' -w \""width"\" -h \""height"\" -t \" '$type'\" -s \""$2"\
        " > "output);
    } else {
      system("./run.sh -i '$interval' -w "width" -h "height" -t \" '$type'\" -s \""$2"\" -e \""
        $3"\" > "output);
    }
  }
}
'
```

B   Output Evidence

B.1   Breadth First Search

```
1        2        3        4
____     ____     ____     ____
____     ____     ---*     ____
---*     ____     ____     ____
abc−     ab*c     abc−     abc*

5        6        7        8
____     ____     ____     ____
____     ____     ____     ____
--*-     --*-     --*-     ____
abc−     ab−c     abc−     a*bc

9        10       11       12
---*     ____     ____     ____
____     ____     --*-     ____
____     ---*     ____     ---*
abc−     abc−     abc−     abc−

13       14       15       16
____     ____     ____     ____
____     --*-     ____     ____
____     ____     ---*     --c-
ab*c     abc−     abc−     ab*−
```

B.2   Depth First Search

```
1        2        3        4
____     ____     ____     ____
____     ____     ____     ____
____     ____     ---*     --*-
ab*c     abc*     abc−     abc−

5        6        7        8
____     ____     ____     ____
____     ____     ____     ____
--c-     --c-     --c-     --c-
ab*−     ab−*     ab*−     ab−*

9        10       11       12
____     ____     ____     ____
____     ____     ____     ---*
--c-     --c-     --c*     --c-
ab*−     ab−*     ab——     ab——

13       14       15       16
____     ____     ____     ____
--*-     --c-     --c-     -*c-
--c-     --*-     -*--     ____
ab——     ab——     ab——     ab——
```

B.3   Iterative Deepening Search

```
Depth increased: 1
1       2
_____   _____

_____   _____

_____   ---*
ab*c    abc-

Depth increased: 2
3       4       5       6
_____   _____   _____   _____

_____   _____   _____   _____

_____   _____   _____   --*-
ab*c    abc*    a*bc    ab-c


7       8       9       10
_____   _____   _____   _____

_____   _____   _____   ---*

---*    _____   --*-    _____
abc-    abc*    abc-    abc-

Depth increased: 3
11      12      13      14
_____   _____   _____   _____

_____   _____   _____   _____

---*    _____   ---*    _____
abc-    abc*    abc-    ab*c


15      16      17
_____   _____   _____

_____   _____   _____

--*-    --c-    -*--
abc-    ab*-    abc-
```

B.4   A* Heuristic Search

```
1       2       3       4
_____   _____   _____   _____

_____   _____   _____   ---*

---*    _____   --*-    _____
abc-    ab*c    abc-    abc-


5       6       7       8
_____   _____   _____   _____

_____   _____   _____   _____

_____   _____   -*--    -b--
abc*    abc*    abc-    a*c-


9       10      11      12
_____   _____   _____   _____

_____   _____   _____   _____

-b--    -b--    -b*-    -b--
ac*-    *ac-    ac--    ac-*


13      14      15      16
---*    _____   _____   _____

_____   _____   _____   _____

_____   -b--    ---*    ---*
abc-    ac*-    abc-    abc-
```

C   Example Output

C.1   Breadth First Search

Running Search :: preRun
Running Search :: runSearch

```
Expanded  Nodes:         29,378     Elapsed  Time  [00:01]     Used  Memory:        58MB
Expanded  Nodes:         71,154     Elapsed  Time  [00:02]     Used  Memory:        77MB
Expanded  Nodes:        124,017     Elapsed  Time  [00:03]     Used  Memory:       130MB
Expanded  Nodes:        166,634     Elapsed  Time  [00:04]     Used  Memory:       120MB
Expanded  Nodes:        236,683     Elapsed  Time  [00:05]     Used  Memory:       167MB
Expanded  Nodes:        288,347     Elapsed  Time  [00:06]     Used  Memory:       331MB
Expanded  Nodes:        336,500     Elapsed  Time  [00:07]     Used  Memory:       286MB
Expanded  Nodes:        398,896     Elapsed  Time  [00:08]     Used  Memory:       282MB
Expanded  Nodes:        458,833     Elapsed  Time  [00:09]     Used  Memory:       462MB
Expanded  Nodes:        496,093     Elapsed  Time  [00:10]     Used  Memory:       355MB
Expanded  Nodes:        574,817     Elapsed  Time  [00:11]     Used  Memory:       603MB
Expanded  Nodes:        625,648     Elapsed  Time  [00:12]     Used  Memory:       528MB
Expanded  Nodes:        687,264     Elapsed  Time  [00:13]     Used  Memory:       479MB
Expanded  Nodes:        765,974     Elapsed  Time  [00:14]     Used  Memory:       724MB
Expanded  Nodes:        819,442     Elapsed  Time  [00:15]     Used  Memory:       644MB
Expanded  Nodes:        909,066     Elapsed  Time  [00:16]     Used  Memory:       670MB
Expanded  Nodes:        998,278     Elapsed  Time  [00:17]     Used  Memory:       692MB
Expanded  Nodes:      1,105,581     Elapsed  Time  [00:18]     Used  Memory:       765MB
Expanded  Nodes:      1,179,148     Elapsed  Time  [00:19]     Used  Memory:       981MB
Expanded  Nodes:      1,295,061     Elapsed  Time  [00:20]     Used  Memory:     1,079MB
Expanded  Nodes:      1,370,995     Elapsed  Time  [00:21]     Used  Memory:     1,054MB
Expanded  Nodes:      1,440,684     Elapsed  Time  [00:22]     Used  Memory:     1,011MB
Expanded  Nodes:      1,547,352     Elapsed  Time  [00:23]     Used  Memory:     1,074MB
Expanded  Nodes:      1,654,448     Elapsed  Time  [00:24]     Used  Memory:     1,138MB
Expanded  Nodes:      1,765,672     Elapsed  Time  [00:25]     Used  Memory:     1,213MB
Expanded  Nodes:      1,856,803     Elapsed  Time  [00:26]     Used  Memory:     1,492MB
Expanded  Nodes:      1,930,003     Elapsed  Time  [00:27]     Used  Memory:     1,444MB
Expanded  Nodes:      2,012,078     Elapsed  Time  [00:28]     Used  Memory:     1,435MB
Expanded  Nodes:      2,100,007     Elapsed  Time  [00:29]     Used  Memory:     1,444MB
Expanded  Nodes:      2,212,357     Elapsed  Time  [00:30]     Used  Memory:     1,520MB
Expanded  Nodes:      2,325,635     Elapsed  Time  [00:31]     Used  Memory:     1,596MB
Expanded  Nodes:      2,438,910     Elapsed  Time  [00:32]     Used  Memory:     1,673MB
Expanded  Nodes:      2,549,779     Elapsed  Time  [00:33]     Used  Memory:     1,749MB
Expanded  Nodes:      2,664,120     Elapsed  Time  [00:34]     Used  Memory:     1,826MB
Expanded  Nodes:      2,766,546     Elapsed  Time  [00:35]     Used  Memory:     2,153MB
Expanded  Nodes:      2,849,977     Elapsed  Time  [00:36]     Used  Memory:     2,130MB
Expanded  Nodes:      2,970,872     Elapsed  Time  [00:37]     Used  Memory:     2,230MB
Expanded  Nodes:      3,075,584     Elapsed  Time  [00:38]     Used  Memory:     2,283MB
Expanded  Nodes:      3,178,468     Elapsed  Time  [00:39]     Used  Memory:     2,328MB
Expanded  Nodes:      3,262,110     Elapsed  Time  [00:40]     Used  Memory:     2,314MB
Expanded  Nodes:      3,349,678     Elapsed  Time  [00:41]     Used  Memory:     2,313MB
Expanded  Nodes:      3,450,024     Elapsed  Time  [00:42]     Used  Memory:     2,362MB
Expanded  Nodes:      3,564,528     Elapsed  Time  [00:43]     Used  Memory:     2,439MB
Expanded  Nodes:      3,669,360     Elapsed  Time  [00:44]     Used  Memory:     2,772MB
Expanded  Nodes:      3,775,457     Elapsed  Time  [00:45]     Used  Memory:     2,822MB
Expanded  Nodes:      3,843,688     Elapsed  Time  [00:46]     Used  Memory:     2,761MB
Expanded  Nodes:      3,949,632     Elapsed  Time  [00:47]     Used  Memory:     2,812MB
Expanded  Nodes:      4,040,206     Elapsed  Time  [00:48]     Used  Memory:     2,821MB
Expanded  Nodes:      4,127,753     Elapsed  Time  [00:49]     Used  Memory:     2,821MB
Expanded  Nodes:      4,239,930     Elapsed  Time  [00:50]     Used  Memory:     2,897MB
Expanded  Nodes:      4,352,605     Elapsed  Time  [00:51]     Used  Memory:     2,973MB
Expanded  Nodes:      4,464,891     Elapsed  Time  [00:52]     Used  Memory:     3,049MB
Expanded  Nodes:      4,564,968     Elapsed  Time  [00:53]     Used  Memory:     3,363MB
Expanded  Nodes:      4,651,719     Elapsed  Time  [00:54]     Used  Memory:     3,356MB
Expanded  Nodes:      4,747,240     Elapsed  Time  [00:55]     Used  Memory:     3,380MB
Expanded  Nodes:      4,842,474     Elapsed  Time  [00:56]     Used  Memory:     3,407MB
Expanded  Nodes:      4,941,967     Elapsed  Time  [00:57]     Used  Memory:     3,438MB
Expanded  Nodes:      5,027,298     Elapsed  Time  [00:58]     Used  Memory:     3,433MB
Expanded  Nodes:      5,140,339     Elapsed  Time  [00:59]     Used  Memory:     3,509MB
Expanded  Nodes:      5,252,009     Elapsed  Time  [01:00]     Used  Memory:     3,585MB
Expanded  Nodes:      5,364,664     Elapsed  Time  [01:01]     Used  Memory:     3,661MB
Expanded  Nodes:      5,477,345     Elapsed  Time  [01:02]     Used  Memory:     3,738MB
Expanded  Nodes:      5,568,689     Elapsed  Time  [01:03]     Used  Memory:     4,025MB
Expanded  Nodes:      5,674,481     Elapsed  Time  [01:04]     Used  Memory:     4,082MB
Expanded  Nodes:      5,744,161     Elapsed  Time  [01:08]     Used  Memory:     4,027MB
Expanded  Nodes:      5,812,563     Elapsed  Time  [01:09]     Used  Memory:     3,969MB
Expanded  Nodes:      5,905,378     Elapsed  Time  [01:10]     Used  Memory:     4,260MB
Expanded  Nodes:      5,992,149     Elapsed  Time  [01:11]     Used  Memory:     4,256MB
Expanded  Nodes:      6,067,506     Elapsed  Time  [01:12]     Used  Memory:     4,214MB
```

```
Expanded  Nodes:      6,150,992    Elapsed  Time  [01:13]    Used  Memory:    4,198MB
Expanded  Nodes:      6,263,272    Elapsed  Time  [01:14]    Used  Memory:    4,275MB
Expanded  Nodes:      6,360,382    Elapsed  Time  [01:15]    Used  Memory:    4,581MB
Expanded  Nodes:      6,405,473    Elapsed  Time  [01:16]    Used  Memory:    4,444MB
Expanded  Nodes:      6,488,545    Elapsed  Time  [01:17]    Used  Memory:    4,428MB
Expanded  Nodes:      6,600,002    Elapsed  Time  [01:18]    Used  Memory:    4,505MB
Expanded  Nodes:      6,696,811    Elapsed  Time  [01:19]    Used  Memory:    4,808MB
Expanded  Nodes:      6,758,005    Elapsed  Time  [01:20]    Used  Memory:    4,724MB
Expanded  Nodes:      6,825,442    Elapsed  Time  [01:21]    Used  Memory:    4,658MB
Expanded  Nodes:      6,939,339    Elapsed  Time  [01:22]    Used  Memory:    4,734MB
Expanded  Nodes:      7,051,978    Elapsed  Time  [01:23]    Used  Memory:    4,768MB
Expanded  Nodes:      7,163,761    Elapsed  Time  [01:24]    Used  Memory:    4,844MB
Expanded  Nodes:      7,274,912    Elapsed  Time  [01:25]    Used  Memory:    4,921MB
Expanded  Nodes:      7,313,015    Elapsed  Time  [01:31]    Used  Memory:    5,039MB
Expanded  Nodes:      7,337,856    Elapsed  Time  [01:32]    Used  Memory:    5,118MB
Expanded  Nodes:      7,454,059    Elapsed  Time  [01:33]    Used  Memory:    5,201MB
Expanded  Nodes:      7,539,038    Elapsed  Time  [01:34]    Used  Memory:    5,090MB
Expanded  Nodes:      7,637,113    Elapsed  Time  [01:35]    Used  Memory:    5,106MB
Expanded  Nodes:      7,722,598    Elapsed  Time  [01:36]    Used  Memory:    5,111MB
Expanded  Nodes:      7,833,920    Elapsed  Time  [01:37]    Used  Memory:    5,187MB
Expanded  Nodes:      7,873,575    Elapsed  Time  [01:43]    Used  Memory:    5,300MB
Expanded  Nodes:      7,951,934    Elapsed  Time  [01:44]    Used  Memory:    5,264MB
Expanded  Nodes:      8,048,079    Elapsed  Time  [01:45]    Used  Memory:    5,567MB
Expanded  Nodes:      8,137,070    Elapsed  Time  [01:46]    Used  Memory:    5,526MB
Expanded  Nodes:      8,229,954    Elapsed  Time  [01:47]    Used  Memory:    5,542MB
```

==

```
Solution found.
Solution as follows:

0:
————
————
————
abc*

1:
————
————
———*
abc—

2:
————
————
——*—
abc—

3:
————
————
—*——
abc—

4:
————
————
—b——
a*c—

5:
————
————
—b——
*ac—

6:
————
————
*b——
—ac—

7:
————
————
b*——
```

```
−ac−
```

8:
```
−−−−
−−−−
ba−−
−∗c−
```

9:
```
−−−−
−−−−
ba−−
−c∗−
```

10:
```
−−−−
−−−−
ba∗−
−c−−
```

11:
```
−−−−
−−∗−
ba−−
−c−−
```

12:
```
−−−−
−∗−−
ba−−
−c−−
```

13:
```
−−−−
−a−−
b∗−−
−c−−
```

14:
```
−−−−
−a−−
∗b−−
−c−−
```

Start State:
```
−−−−
−−−−
−−−−
abc∗
```

Exit State:
```
−−−−
−a−−
−b−−
−c−−
```

Random Seed:
−2679893794501661041

Nodes Expanded:
8,318,621

C.2   Depth First Search

```
Running  Search :: preRun
Running  Search :: runSearch

Expanded  Nodes :         2,766    Elapsed  Time  [00:00]    Used  Memory :      1MB
Expanded  Nodes :        15,459    Elapsed  Time  [00:00]    Used  Memory :      3MB
Expanded  Nodes :        33,489    Elapsed  Time  [00:00]    Used  Memory :     13MB
Expanded  Nodes :        49,812    Elapsed  Time  [00:00]    Used  Memory :     55MB
Expanded  Nodes :        68,246    Elapsed  Time  [00:01]    Used  Memory :     57MB
Expanded  Nodes :        85,650    Elapsed  Time  [00:01]    Used  Memory :     75MB
```

_____

```
Solution  found .
Solution  as  follows :
Expanded  Nodes :        92,972    Elapsed  Time  [00:01]    Used  Memory :     69MB
0:
____
____
____
abc*

1:
____
____
____
ab*c

2:
____
____
--*-
ab-c

3:
____
____
-*--
ab-c

4:
____
-*--
____
ab-c

5:
____
--*-
____
ab-c

6:
--*-
____
____
ab-c

7:
---*
____
____
ab-c

8:
____
---*
____
ab-c

....
....
....
....

92965:
```

```
————
—a*—
bc——
————
```

92966:
```
————
—a—*
bc——
————
```

92967:
```
————
—a*—
bc——
————
```

92968:
```
————
—a——
bc*—
————
```

92969:
```
————
—a——
bc——
——*—
```

92970:
```
————
—a——
bc——
—*——
```

92971:
```
————
—a——
b*——
—c——
```

92972:
```
————
—a——
*b——
—c——
```

Start State:

```
————
————
————
abc*
```

Exit State:

```
————
—a——
—b——
—c——
```

Random Seed:
−6659880257389911118

Nodes Expanded:
92972

C.3   Iterative Deepening Search

```
Running Search :: preRun
Running Search :: runSearch

Depth increased : 1

Depth increased : 2

Depth increased : 3

Depth increased : 4

Depth increased : 5
Expanded Nodes:            189     Elapsed Time [00:00]     Used Memory:

Depth increased : 6
Expanded Nodes:            870     Elapsed Time [00:00]     Used Memory:

Depth increased : 7
Expanded Nodes:          4,122     Elapsed Time [00:00]     Used Memory:

Depth increased : 8
Expanded Nodes:         15,569     Elapsed Time [00:00]     Used Memory:

Depth increased : 9
Expanded Nodes:         44,117     Elapsed Time [00:01]     Used Memory:     87MB

Depth increased : 10
Expanded Nodes:        162,212     Elapsed Time [00:02]     Used Memory:    140MB

Depth increased : 11
Expanded Nodes:        302,767     Elapsed Time [00:03]     Used Memory:    143MB
Expanded Nodes:        465,493     Elapsed Time [00:04]     Used Memory:     32MB

Depth increased : 12
Expanded Nodes:        649,869     Elapsed Time [00:05]     Used Memory:      4MB
Expanded Nodes:        835,458     Elapsed Time [00:06]     Used Memory:    138MB
Expanded Nodes:      1,018,926     Elapsed Time [00:07]     Used Memory:    104MB
Expanded Nodes:      1,167,648     Elapsed Time [00:08]     Used Memory:    116MB
Expanded Nodes:      1,342,004     Elapsed Time [00:09]     Used Memory:     57MB
Expanded Nodes:      1,524,357     Elapsed Time [00:10]     Used Memory:     25MB
Expanded Nodes:      1,699,719     Elapsed Time [00:11]     Used Memory:    121MB

Depth increased : 13
Expanded Nodes:      1,824,296     Elapsed Time [00:12]     Used Memory:     71MB
Expanded Nodes:      1,966,793     Elapsed Time [00:13]     Used Memory:     50MB
Expanded Nodes:      2,162,291     Elapsed Time [00:14]     Used Memory:     57MB
Expanded Nodes:      2,348,533     Elapsed Time [00:15]     Used Memory:     37MB
Expanded Nodes:      2,540,465     Elapsed Time [00:16]     Used Memory:     34MB
Expanded Nodes:      2,732,738     Elapsed Time [00:17]     Used Memory:     31MB
Expanded Nodes:      2,920,399     Elapsed Time [00:18]     Used Memory:     16MB
Expanded Nodes:      3,103,913     Elapsed Time [00:19]     Used Memory:    137MB
Expanded Nodes:      3,295,237     Elapsed Time [00:20]     Used Memory:    131MB
Expanded Nodes:      3,489,259     Elapsed Time [00:21]     Used Memory:    135MB
Expanded Nodes:      3,667,882     Elapsed Time [00:22]     Used Memory:     89MB
Expanded Nodes:      3,835,332     Elapsed Time [00:23]     Used Memory:     10MB
Expanded Nodes:      4,022,254     Elapsed Time [00:24]     Used Memory:    141MB
Expanded Nodes:      4,214,688     Elapsed Time [00:25]     Used Memory:    141MB
Expanded Nodes:      4,403,704     Elapsed Time [00:26]     Used Memory:    127MB
Expanded Nodes:      4,595,737     Elapsed Time [00:27]     Used Memory:    124MB
Expanded Nodes:      4,787,754     Elapsed Time [00:28]     Used Memory:    122MB
Expanded Nodes:      4,969,529     Elapsed Time [00:29]     Used Memory:     85MB
Expanded Nodes:      5,147,124     Elapsed Time [00:30]     Used Memory:     44MB
Expanded Nodes:      5,341,106     Elapsed Time [00:31]     Used Memory:     40MB
Expanded Nodes:      5,515,405     Elapsed Time [00:32]     Used Memory:    133MB
Expanded Nodes:      5,676,501     Elapsed Time [00:33]     Used Memory:     40MB

Depth increased : 14
Expanded Nodes:      5,853,237     Elapsed Time [00:34]     Used Memory:    133MB
Expanded Nodes:      6,045,152     Elapsed Time [00:35]     Used Memory:    129MB
Expanded Nodes:      6,228,239     Elapsed Time [00:36]     Used Memory:     99MB
Expanded Nodes:      6,396,529     Elapsed Time [00:37]     Used Memory:     24MB
════════════════════════════════

Solution found .
```

Solution as follows:

0:
____
____
____
abc*

1:
____
____
---*
abc-

2:
____
____
--*-
abc-

3:
____
____
-*--
abc-

4:
____
____
-b--
a*c-

5:
____
____
-b--
*ac-

6:
____
____
*b--
-ac-

7:
____
____
b*--
-ac-

8:
____
____
ba--
-*c-

9:
____
____
ba--
-c*-

10:
____
____
ba*-
-c--

11:
____
--*-
ba--
-c--

12:
____

```
-*--
ba—
-c—

13:
____

-a—
b*--
-c—

14:
____

-a—
*b—
-c—
```

================================

Start State:

```
____
____
____
abc*
```

================================

Exit State:

```
____
-a—
-b—
-c—
```

================================

Random Seed:
-2736763515611179222

================================

Nodes Expanded:
6430966

================================

Max Iterative Depth:
14

C.4 A* Heuristic Search

Running Search::preRun
Running Search::runSearch

Expanded Nodes:	0	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	38	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	170	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	427	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	741	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	972	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	1,449	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	2,042	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	2,403	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	2,627	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	3,200	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	3,759	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	4,309	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	4,706	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	5,089	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	5,480	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	5,833	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	6,215	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	6,609	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	7,013	Elapsed Time [00:00]	Used Memory:	
Expanded Nodes:	7,438	Elapsed Time [00:01]	Used Memory:	93MB
Expanded Nodes:	7,548	Elapsed Time [00:01]	Used Memory:	
Expanded Nodes:	8,169	Elapsed Time [00:01]	Used Memory:	
Expanded Nodes:	8,785	Elapsed Time [00:01]	Used Memory:	
Expanded Nodes:	8,997	Elapsed Time [00:01]	Used Memory:	

==

Solution found.
Solution as follows:

0:
————
————
————
abc*

1:
————
————
———*
abc—

2:
————
————
——*—
abc—

3:
————
————
—*——
abc—

4:
————
————
—b——
a*c—

5:
————
————
—b——
*ac—

6:
————
————
*b——
—ac—

7:
____
____
b*——
—ac—

8:
____
____
ba——
—*c—

9:
____
____
ba——
—c*—

10:
____
____
ba*—
—c——

11:
____
——*—
ba——
—c——

12:
____
—*——
ba——
—c——

13:
____
—a——
b*——
—c——

14:
____
—a——
*b——
—c——

════════════════════════════
Start State:

____
____
____
abc*

════════════════════════════
Exit State:

____
—a——
—b——
—c——

════════════════════════════
Random Seed:
7965083047300516107
════════════════════════════
Nodes Expanded:
9,491
════════════════════════════

D   Graphs

D.1   2x2 Grid, Complexity vs Nodes Evaluated
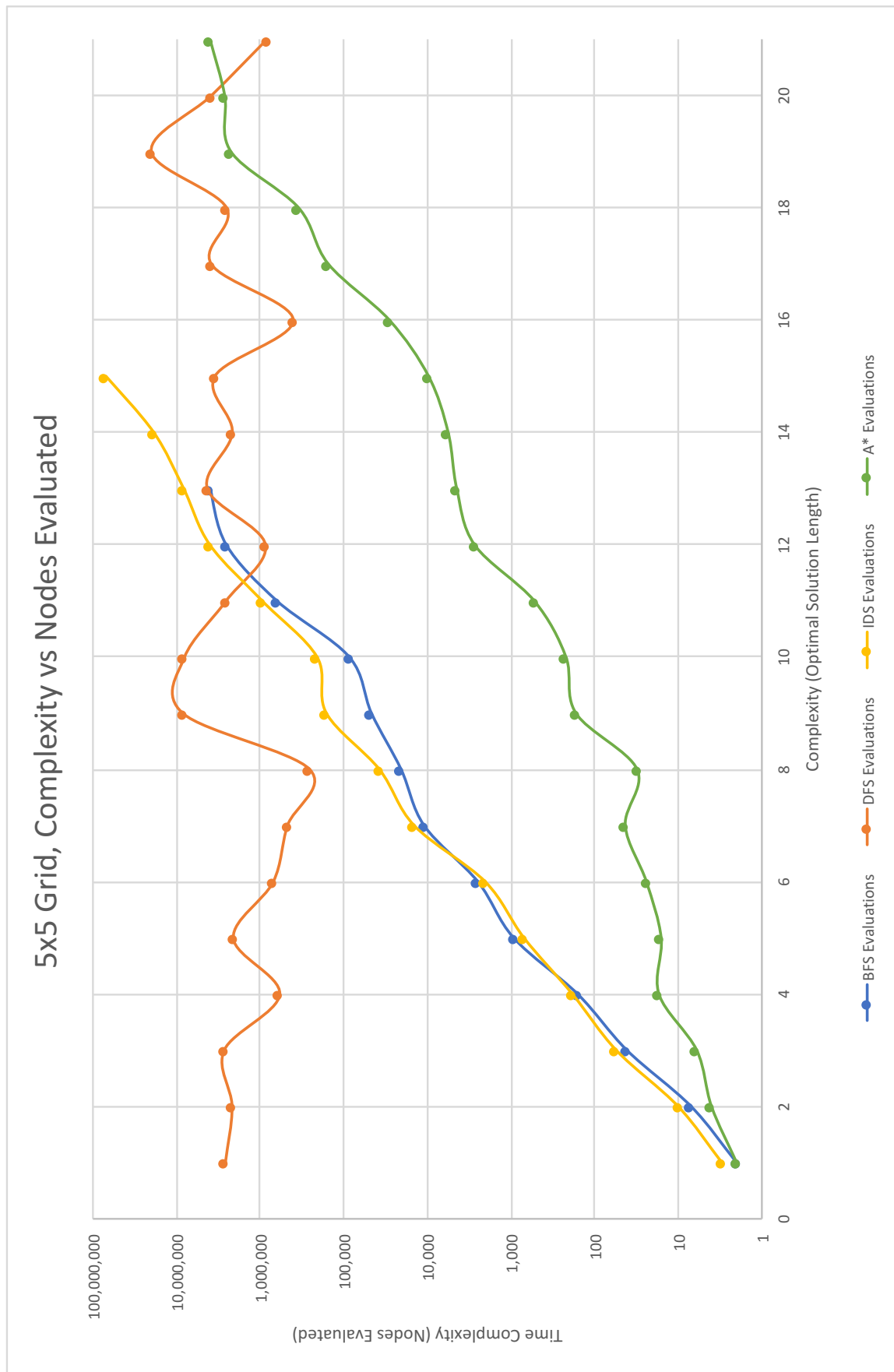
D.2   3x3 Grid, Complexity vs Nodes Evaluated
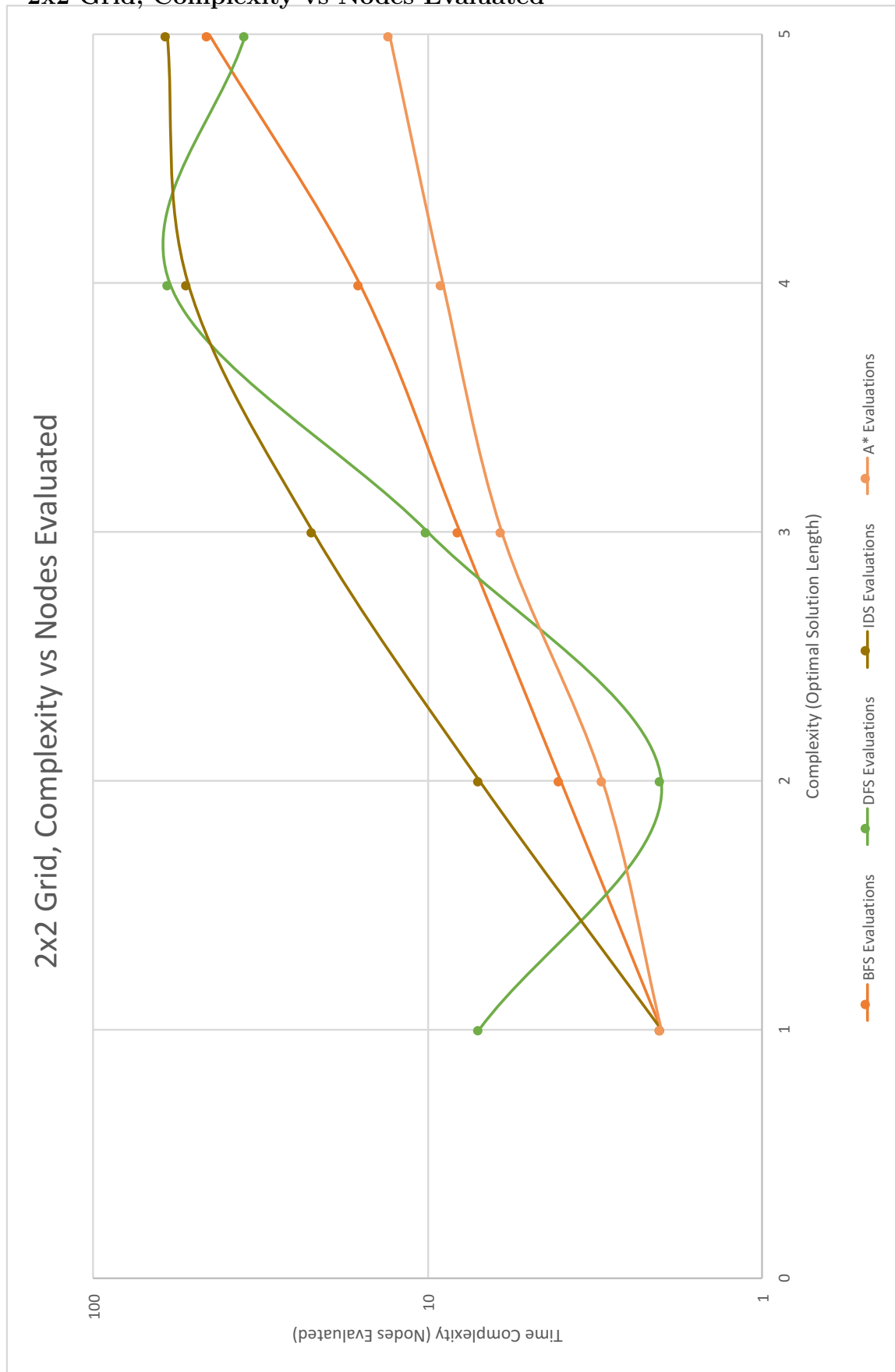
3x3 Grid, Complexity vs Nodes Evaluated

D.3   4x4 Grid, Complexity vs Nodes Evaluated
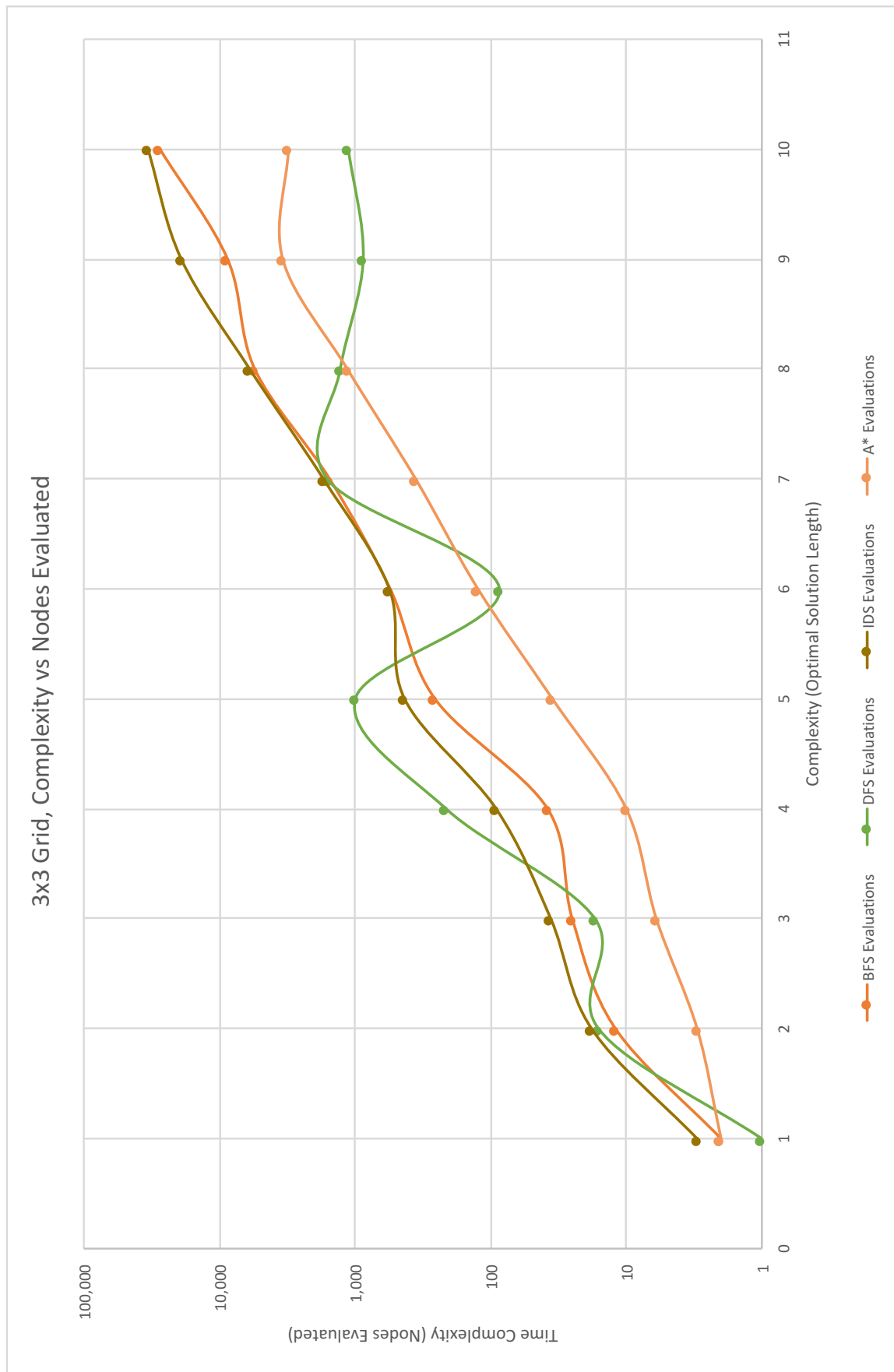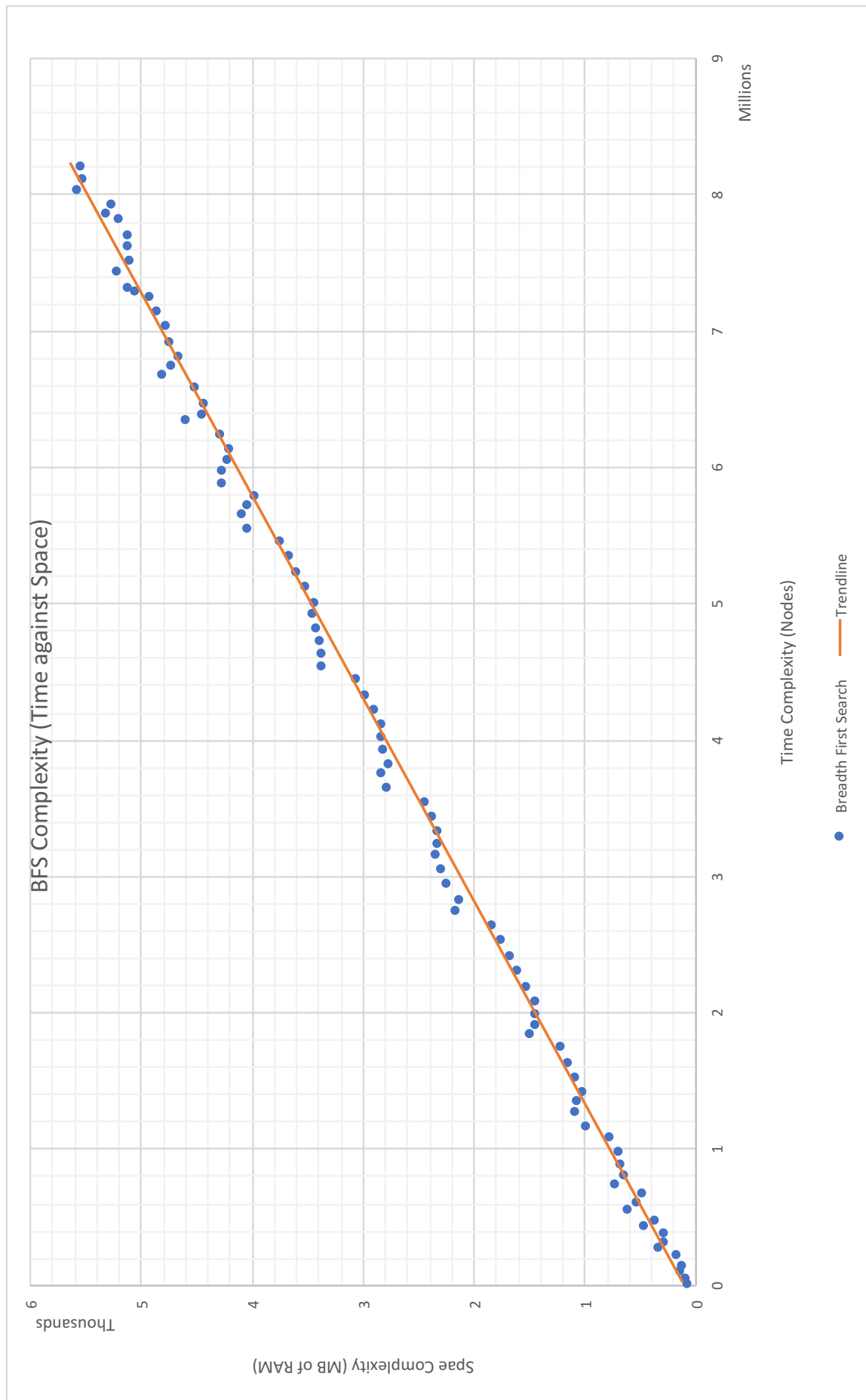

4x4 Grid, Complexity vs Nodes Evaluated

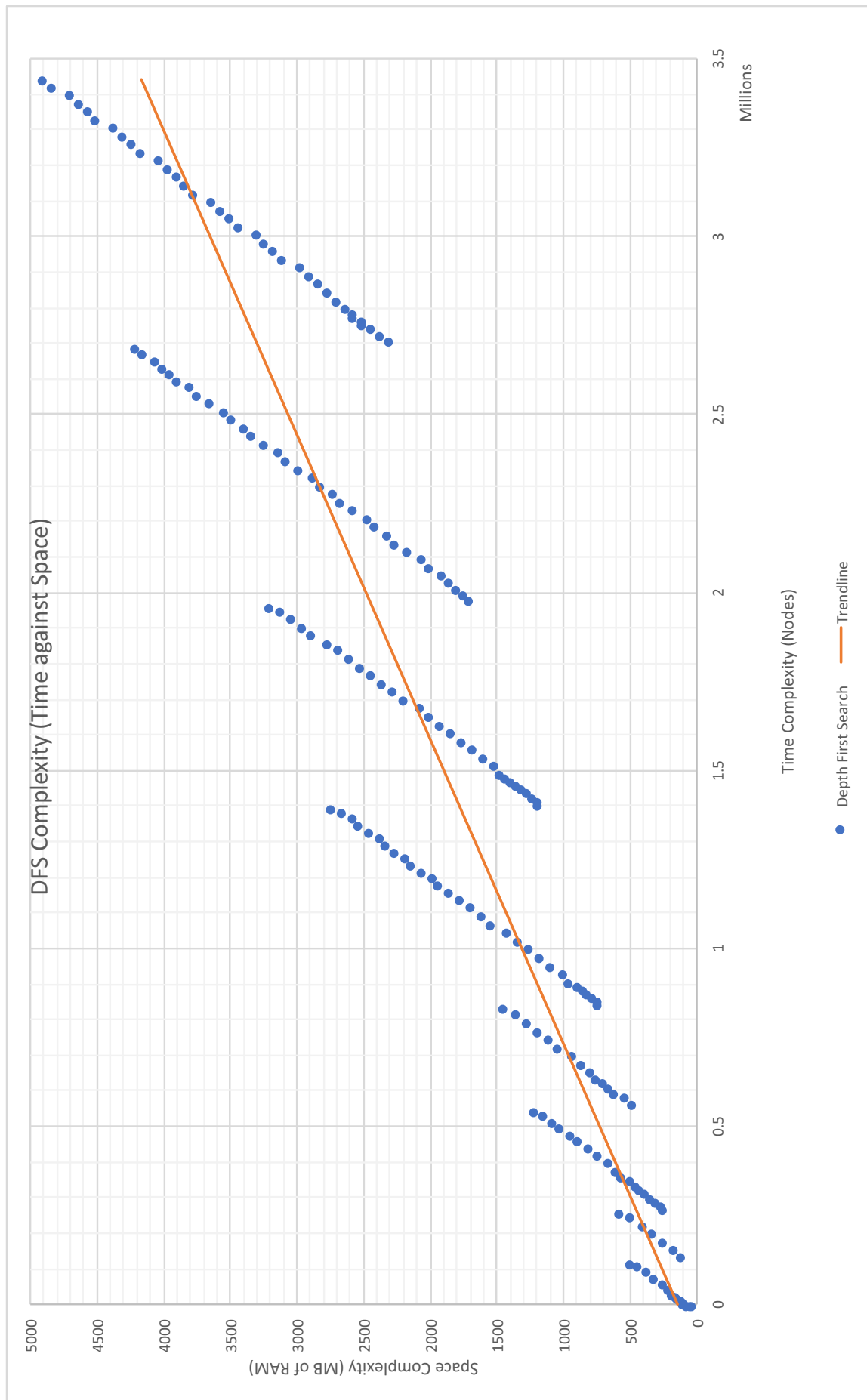## D.4   5x5 Grid, Complexity vs Nodes Evaluated
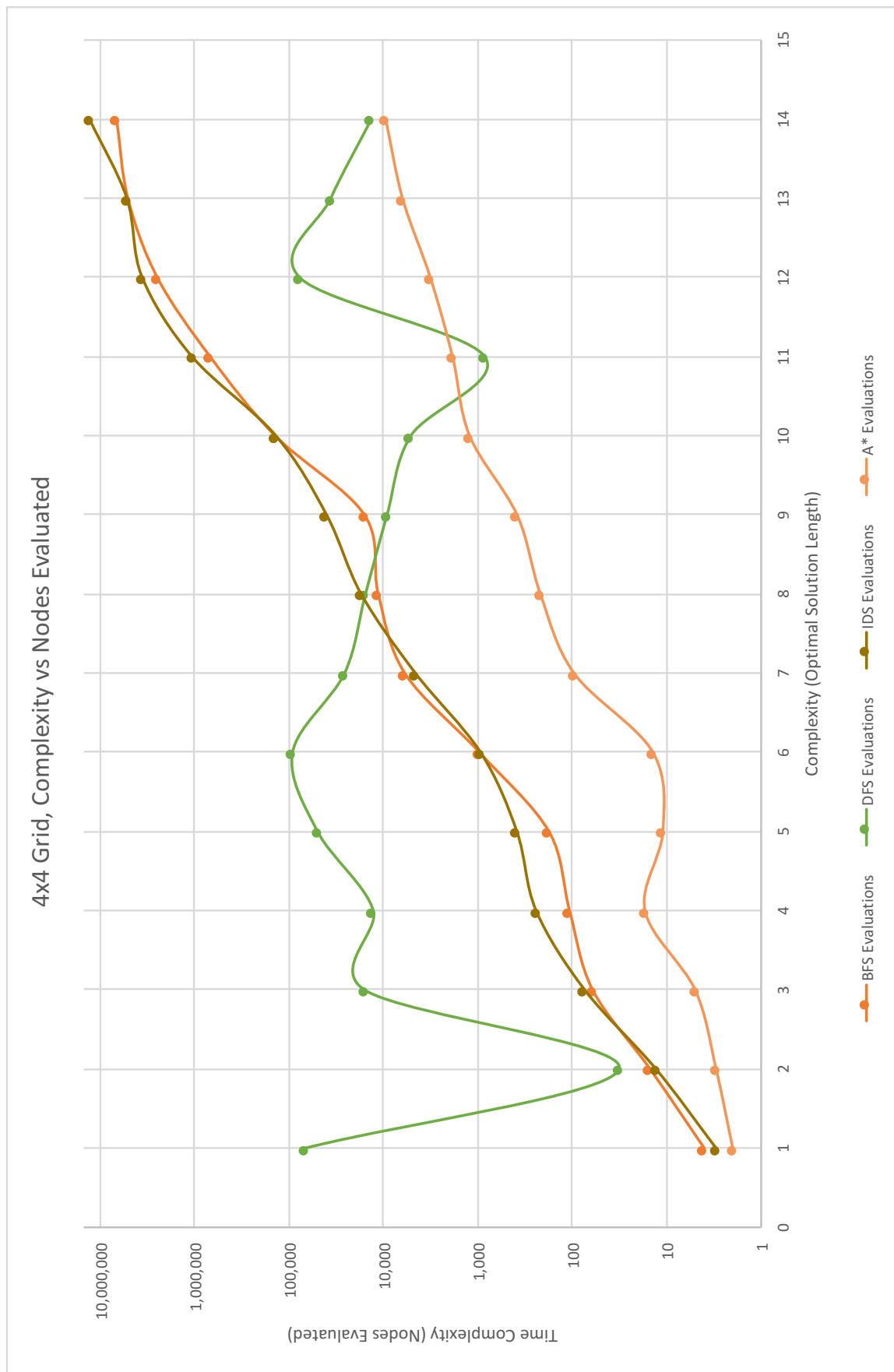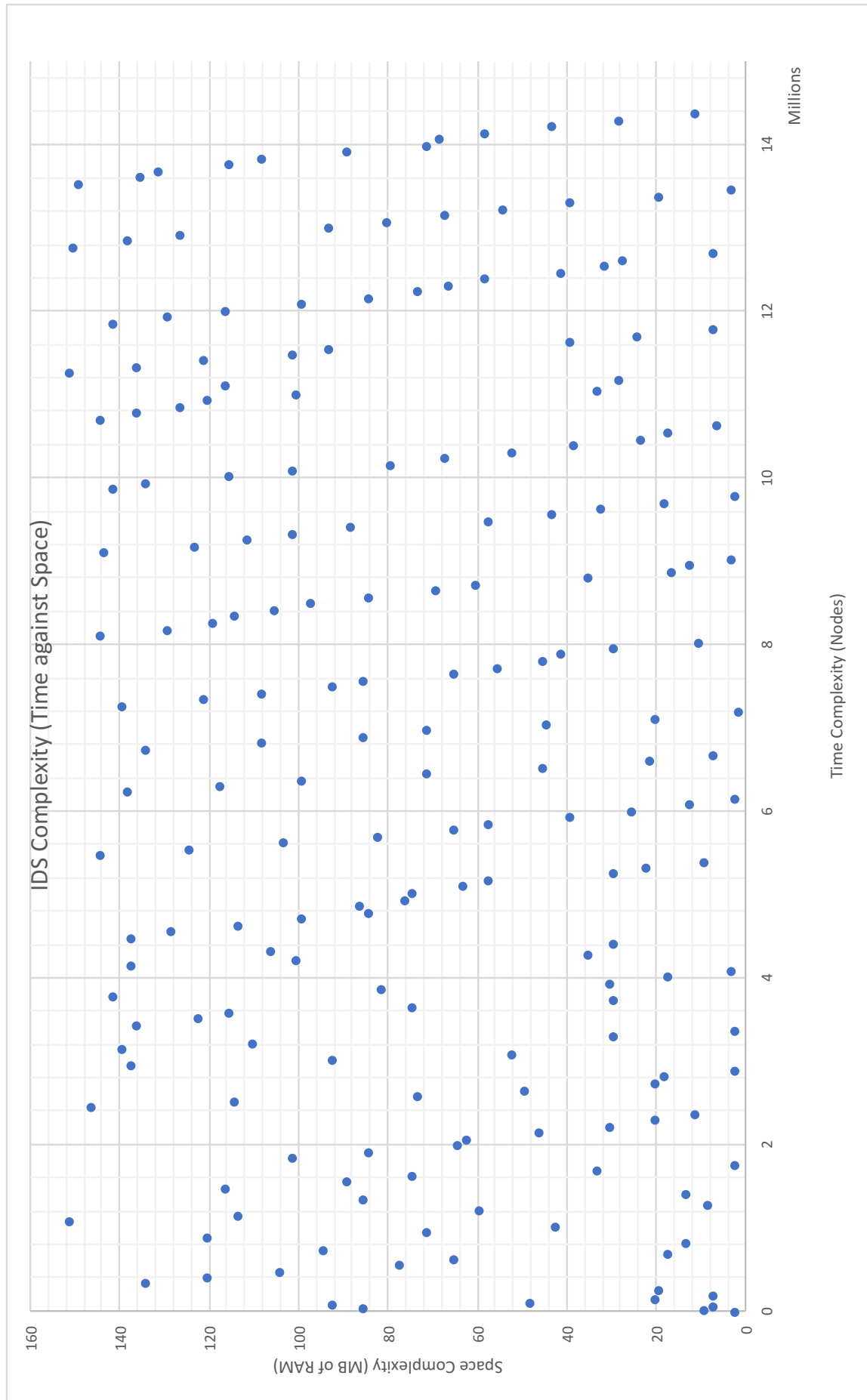

5x5 Grid, Complexity vs Nodes Evaluated

D.5 BFS Complexity (Time vs Space)

D.6   DFS Complexity (Time vs Space)

D.7   IDS Complexity (Time vs Space)

IDS Complexity (Time against Space)

Space Complexity (MB of RAM)

Time Complexity (Nodes)

Millions

D.8   A* Complexity (Time vs Space)

A* Complexity (Time against Space)