

UNIVERSITY OF SOUTHAMPTON

COMP2208: Intelligent Systems

COMPARISON OF SEARCH METHODS

Huw Jones
27618153
hcbj1g15@soton.ac.uk

November 29, 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          Sets the grid height.
    --help            Prints this help message.
-r, --ran [LONG]      Specifies the seed used for the pseudo-random number.
-s, --start [STATE]   Specifies the start state
-t, --type            Specifies the search type:
                        BFS - Breadth First Search
                        DFS - Depth First Search
                        IDS - Iterative Deepening Search
                        A* - A* Heuristic Search
-w, --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 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
```

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

Thanks to my framework, it was trivial to control the complexity of puzzles to solve. I created a format for specifying start/exit states.

4 Extras & Limitations

5 References

Appendices

A Code

A.1 BlocksWorld.java

```
import blocksworld.Grid;
import blocksworld.GridController;
import blocksworld.Pair;
import blocksworld.Position;
import blocksworld.exceptions.InvalidPositionException;
import blocksworld.search.*;

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)");
            }
            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] \t Specifies _exit _state.");
    System.out.println("--h, --height [HEIGHT] \t Sets _the _grid _height.");
    System.out.println("-----help \t \t Prints _this _help _message.");
    System.out.println("--i, --interval [TIME] \t Sets _the _refresh _interval _ (in _ms) _for _
    monitoring _search _status.");
    System.out.println("--r, --ran [STATE] \t Specifies _the _seed _used _for _the _pseudo-random
    _number.");
    System.out.println("--s, --start [STATE] \t Specifies _the _start _state");
    System.out.println("--t, --type [TYPE] \t Specifies _the _search _type: \r\n\t\t BFS _
    Breadth_First_Search \r\n\t\t DFS _ Depth_First_Search \r\n\t\t IDS _ Iterative _
    Deepening_Search \r\n\t\t A* _ A*_Heuristic_Search");
    System.out.println("--w, --width [WIDTH] \t Sets _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);
}

```

```

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 == '*') {
                        g.placeAgent(x, i - 2);
                    }
                } 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

```

package blocksworld;

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 {
        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 {
        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
 * @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

```

package blocksworld;

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 {
        grid.placeAgent(x, y);
        return grid;
    }

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

```

```

        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

```
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

```
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

```
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

```
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

```
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

```
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

```

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();
    }
}

```



```

/**
 * 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

```

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

```

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

```

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);
                    }
                } catch (InvalidDirectionException e) {
                    // This should not happen as we've already checked the direction is valid
                }
            }
        }
    }
}

```

```

        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

```

package blocksworld.search;

import blocksworld.*;
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 e) {
            e.printStackTrace();
        }
    }

    /**
     * Builds the default 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();
        }
    }
}

```

```

}

/**
 * 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
     * @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 / 1000000000L;
                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("%02d", minutes).replace('_', '0'),
                    String.format("%02d", seconds).replace('_', '0'),
                    NumberFormat.getNumberInstance(Locale.getDefault()).format(memory)));
                try {
                    Thread.sleep(Search.this.getRefreshTime());
                } catch (InterruptedException e) {
                }
            }
        }
    }
}

```

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

=====