

**Department of Computing**

**Algorithms and Data Structures**

**(55-508810-AF-202425)**

**Name: J. A. Rowden**

**Student ID: 33004175**

**Table of Contents**

[1. Project 1 3](#_Toc178928233)

[1.1 Computational Complexity 3](#_Toc178928234)

[1.2 Observation of rate of growth 3](#_Toc178928235)

[1.3 Incorporation of formative feedback 4](#_Toc178928236)

[2. Project 2 5](#_Toc178928237)

[2.1 Algorithms in an Algorithmic Language, such as ADL 5](#_Toc178928238)

[2.2 Data Structures 7](#_Toc178928239)

[2.3 Software and its Presentation, including testing (and video link) 7](#_Toc178928240)

[2.4 Descriptive Report, including artefacts 10](#_Toc178928241)

[2.4.1 Transitioning algorithms to implementation 10](#_Toc178928242)

[2.4.2 Problem-solving strategy 11](#_Toc178928243)

[2.5 Incorporation of formative feedback 13](#_Toc178928244)

[3. Project 3 14](#_Toc178928245)

[3.1 **Data Structure selection** 14](#_Toc178928246)

[3.2 Developed Algorithms 14](#_Toc178928247)

[3.2.1 Algorithm1: Influencers in Unweighted Network 14](#_Toc178928248)

[Time Complexity of Algorithm1: Discuss about the time complexity of Algorithm1 that you have developed. 14](#_Toc178928249)

[3.2.2 Algorithm2: Influencers in Weighted Network 14](#_Toc178928250)

[Time Complexity of Algorithm2: Discuss about the time complexity of Algorithm2 that you have developed. 14](#_Toc178928251)

[3.3 Implementation 15](#_Toc178928252)

[3.3.1 Implementation details 15](#_Toc178928253)

[3.3.2 Publishing your project 15](#_Toc178928254)

[3.3.3 Evaluation 15](#_Toc178928255)

[3.4 Reflection 15](#_Toc178928256)

[3.4.1 Video 15](#_Toc178928257)

[3.4.2 Feedback 15](#_Toc178928258)

[4. Project 4 16](#_Toc178928259)

[4.1 Introduction to Bin Packing Problem 16](#_Toc178928260)

[4.2 Random Mutation Hill Climbing 16](#_Toc178928261)

[4.3 Algorithm Design 16](#_Toc178928262)

[4.3.1 Solution Representation 16](#_Toc178928263)

[4.3.2 Fitness Function Analysis 16](#_Toc178928264)

[4.3.2 Small Change Strategy 16](#_Toc178928265)

[4.3.4 Data Structure 16](#_Toc178928266)

[4.3.5 Algorithm Pseudocode 16](#_Toc178928267)

[4.3.6 Experiment Strategy 16](#_Toc178928268)

[4.4 Results 16](#_Toc178928269)

[4.5 Testing 16](#_Toc178928270)

[4.6 Software and its Presentation, including testing (and video link) 16](#_Toc178928271)

[4.7 Incorporation of formative feedback 17](#_Toc178928272)

# 1. Project 1

## Computational Complexity

A close up of text

Description automatically generated

*Outer Loop*

*Middle Loop*

*Inner Loop*

**Find big O of above algorithm:**

**Points to note:**

* Print statement runs k number of times for every time the inner loop runs
* Inner loop runs j number of times for every time the middle loop runs
* middle loop runs i number of times for every time the outer loop runs
* outer loop runs n number of times

therefore:

=

=

Subbing in this:

Factoring out constants:

Separating into two series

Dealing with each series individually:

=

Subbing back in and simplifying:

)

)

)

)

)

**n3 is dominant term therefore the algorithm has a Big O of O(n3)**

## Observation of rate of growth

Let the two functions be known as:

F(n) = n2

G(n) = 2n/4

To observe the rate of growth of these functions so I can comment on them I have devised 2 methods.

Method 1: table of values

|  |  |  |
| --- | --- | --- |
| n | f(n) | g(n) |
| 1 | 1 | 0.5 |
| 5 | 25 | 8 |
| 10 | 100 | 256 |
| 15 | 225 | 8192 |
| 20 | 400 | 262144 |
| 25 | 625 | 8388608 |
| 30 | 900 | 2.68E+08 |

Method 2: graphics calculator application

*Using* [*https://www.geogebra.org/calculator*](https://www.geogebra.org/calculator) *:*

*A screenshot of a graph

Description automatically generated*

*Note: x is used as a substitute for n and is limited to the positive scale as negative complexity does not need to be considered.*

Analysis

As shown best by the graph g(n) is better for small values of n =1 up till n = 8 (see intersect point B). Note I’m ignoring the first intercept point A as data sets normally consist of 1 or more items, not of ~ 0.062 of an item. However, past a data set size of 8, as best shown by the table if values f(n) is significantly better than g(n) as the difference between f(n) and g(n) grows exponentially large.

## 1.3 Incorporation of formative feedback

|  |  |  |  |
| --- | --- | --- | --- |
| Date and Time | Name of the Tutor | Feedback of the tutor | How the comment is addressed |
| 04/10/24 ~ 10:40 | Mehemet | Excellent work | N/A |
|  |  |  |  |

# 2. Project 2

## 2.1 Algorithms in an Algorithmic Language, such as ADL

I have designed two separate algorithms and corresponding data structures (A&DS) to solve this problem, based on depth first and breadth first traversals of graphs to avoid confusion on which algorithms go with which data structures I will list common A&DS and then under separate headings the A&DS for each approach.

Common Data Structures.

**2D array – used for Maze Representation:**

* The maze doesn’t change, only accessed so the lack of dynamics doesn’t matter
* Space efficient
* Easily accessed
* *Note:* Though I represented the maze using a 2D array I treated the maze more like a graph in both solutions with adjacent 0s being connected and ignoring the 1s.

**Endless Stack – used to store chain of visited nodes/positions**

* Custom data type
* A dynamic stack as I would not be able to tell how big it needs to be before running the algorithm, and this size would change with different mazes and algorithms.
* Used as I wanted a data structure that preserved the order on which items were added to it so the route would not be compromised
* **Functions:**
  + **isEmpty()**
  + **push() – adds an item to the top of the stack**
  + **pop() – returns and removes the top item on the stack**
  + **convertToList() – returns in list form for display**
  + **Count() – returns integer value of number of item in stack.**

Common Algorithms

**InRange – checks if a position that algorithm was looking at was inside the map.**

* Used to avoid errors
* Used to recognize the exit.
* Implemented in a parent class of both algorithms’ classes.
* *Note:* this was implemented during development after receiving feedback from lecturer hence no ADL

Depth First Data Structures

**Array – Used for Positions/ positions**

* The maze doesn’t change dimensions, so the lack of dynamics doesn’t matter
* Space efficient compared to an object approach
* Easily accessed

Depth First Algorithms

In a low-level pseudo code:

A screenshot of a computer code

Description automatically generated Breadth First Data Structures

**Queue – Used to store nodes to visit next**

* Dynamic for same reasons as Endless stack
* Use of First in First out to implement a breadth first solution.

**Node**

* Custom data structure
* Creates a pseudo linked list with its recursive definition
* **Used to:**
  + store positions of the nodes the algorithm will traverse to
  + to store positions of the nodes the algorithm will traverse to
  + store if end found
* **Has**
  + Array to store position of the node
  + Node to point to the node this was node was traversed to from
  + Boolean value to signify if this node is adjacent to the exit.

Breadth First Algorithms

In a high level pseudo code :

A screenshot of a computer program

Description automatically generated

## 2.3 Software and its Presentation, including testing (and video link)

[*https://youtu.be/nR\_NoYhzUIs*](https://youtu.be/nR_NoYhzUIs)

*In your video recording, you will need to demonstrate how you tested the software and what results you obtained using the acceptance tests outlined in this section, e.g.,*

Acceptance tests

| N.O | Acceptance test- *Green = Expected route; Blue = start Red = exit* | Expected result | Depth first Result | Breadth first Result |
| --- | --- | --- | --- | --- |
| 1 | 1,0,1,1,1,1,1,1  1,0,0,0,0,0,0,1  1,1,1,1,1,1,0,1  1,0,0,0,0,0,0,1  1,0,1,1,1,1,1,1  1,0,0,0,0,0,0,1  1,1,1,1,1,1,0,1  1,0,0,0,0,0,0,1  1,0,1,1,1,1,1,1  1,0,0,0,0,0,0,0  1,1,1,1,1,1,1,1 | (1,0), (1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (2,6), (3,6) (3,5), (3,4), (3,3), (3,2), (3,1), (4,1), (5,1), (5,2), (5,3), (5,4), (5,5), (5,6), (6,6), (7,6) (7,5), (7,4), (7,3), (7,2), (7,1), (8,1), (9,1), (9,2), (9,3), (9,4), (9,5), (9,6), (9,7) | (0,1),(1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (2,6), (3,5), (3,4), (3,3), (3,2), (3,1), (4,1), (5,1), (5,2), (5,3), (5,4), (5,5), (5,6), (6,6), (7,5), (7,4), (7,3), (7,2), (7,1), (8,1), (9,1), (9,2), (9,3), (9,4), (9,5), (9,6), (9,7), | (1,2),(1,3), (1,4), (1,5), (2,6), (3,5), (3,4), (3,3), (3,2), (4,1), (5,2), (5,3), (5,4), (5,5), (6,6), (7,5), (7,4), (7,3), (7,2), (8,1), (9,2), (9,3), (9,4), (9,5), (9,6), (9,7), |
| 2 | 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1  1,0,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1  1,1,0,0,0,1,1,0,1,1,1,0,0,1,1,1,1  1,0,1,1,0,0,0,0,0,1,1,1,0,0,1,1,1  1,1,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1  1,1,0,1,0,0,1,0,1,1,1,1,1,1,1,1,1 1,0,0,1,1,0,1,1,1,0,1,0,0,1,0,1,1  1,0,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1  1,0,0,1,1,0,1,1,0,1,1,1,1,1,1,0,1  1,1,0,0,0,1,1,0,1,1,0,0,0,0,0,0,1  1,0,0,1,1,1,1,1,0,0,0,1,1,1,1,0,1  1,0,1,0,0,1,1,1,1,1,0,1,1,1,1,0,0 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1 | (1,1), (2,2), (1,3), (2,4), (2,5), (2,6) ,(1,6), (1,7), (1,8) (2,8), (2,9), (3,9), (4,9), (5,8), (6,7), (7,7), (8,8), (7,9), (8,10), (9,10), (10,10), (10,9), (11,9), (12,9), (13,9), (14,9), (15,9), (15,10), (15,11), (16,11) | (1,1), (2,2), (3,1), (4,2), (5,2), (6,1), (6,2), (7,1), (8,1), (8,2), (9,2), (9,3), (9,4), (8,5), (7,6), (7,7), (8,8), (9,7), (10,8), (10,9), (9,10), (9,11), (9,12), (9,13), (9,14), (8,15), (9,15), (10,15), (11,15), (11,16) | (2,2), (3,1), (4,2), (5,2), (6,1), (7,1), (8,2), (9,3), (9,4), (8,5), (7,6), (7,7), (8,8), (9,7), (10,8), (10,9), (9,10), (9,11), (9,12), (9,13), (9,14), (10,15), (11,16), |

Comments on acceptance tests

**Acceptance test 1**

* All routes are correct but different
* Because of the way arrays work in C# the x and y of the coordinates are flipped
* When I made my ‘expected route’ I forgot that the algorithms could move diagonally. Note how depth first only does this in places like – using (y,x) –(2,6) to (3,5) whereas breadth first always cuts the corner. This is because of the way the algorithms work. They both start top left and work their way across left to right top to bottom. Depth first goes to the first zero it sees, hence why it only sometimes uses the diagonal whereas breadth first goes to all, and the shortest route is found after hence the consistent corner cutting. This would suggest that breadth first is better at finding the shortest route, but depth first is better at finding a route fastest

## 2.4 Descriptive Report, including artefacts

### 2.4.1 Transitioning algorithms to implementation

During implementation I changed from the ADL significantly to simplify the program. I realised for both solutions I didn’t need a look around function if I was just going to loop through again. So instead of ‘looking around’ then processing I process as I ‘look around’ this reduces the time complexity of both algorithms and means there are less function calling.

I also decided to wrap both solution’s code in a class so I can take advantage of encapsulation and inheritance. I created a common class routingAlgorithm that stores the inRange function and originally the look around function. inRange, is a function that I realised need to make because of how C# handles out of range accessing that I overlooked when designing the algorithms in ADL.

Depth first has a findRoutePt2 function as I wanted to have the solutions return the same data structure but still wanted to use and Endless stack in the recursion element so in the Depth first code findRoute acts as wrapper function that formats the output of the recursive findRoutePt2.

### 2.4.2 Problem-solving strategy

When considering this problem, I saw each 0 in the maze as a node of a graph, and thus I was able to use Algorithms associated with graphs. Despite these algorithms I decided to store the maze as a matrix as this was the closest representation to the mazes given and my relative experience at using matrices to graphs in C#. When designing the two algorithms, I focused on three things: time complexity, space complexity and the length of the route found.

Solution 1 Depth first:

I designed this algorithm to be the leanest and fastest so low time and space complexity; the route’s length wasn’t important just that it worked. I would imagine that this style of algorithm would be used in a game pathing situations where the speed of the operation was a priority to ensure the program ran smoothly.

It could almost be called a greedy algorithm as it takes the first path it sees. It ‘looks around’ by using two nested for loops to first check the three nodes above, then the three nodes on the same row as it then the three nodes below.

A black background with numbers and red arrows

Description automatically generated

*Above: diagram of how the algorithm ‘looks around’*

I use recursion to handle the repetitive nature, in this way little code is used implementing a loop and little space is wasted storing information that isn’t used. When a node is traversed to, it pushes it to the Endless Stack. When the end node is found it returns the Endless Stack; if it can’t find a node to traverse to that hasn’t been traversed to yet, it returns null.

**time complexity (worst case):** O(n)

**time complexity (best case):** O(s)

Where:

n = number of nodes in the matrix.

s = the number of nodes in the most optimal route.

**Explanation:**

Though I do use nested for loops they are of constant time complexity as combined they run 9 times for every node visited, the only loop that depends on the input size is the loop created by the recursion - so for every node visited - which in the worst case would be all nodes in the matrix.

Solution 2 Breadth first:

I designed this algorithm to be one that prioritised the shortest route regardless of the time or space complexity. I would imagine that this style of algorithm would be used in a mapping solution like Google Maps where the route distance is the priority.

The algorithm works by ‘looking around’ the current node (in the same way as depth first) and adding any possible traversable nodes to a queue if they haven’t already been visited. To do this I employ a custom data structure node to store the node’s position in the matrix, whenever it’s the end node, and the node it would be visited by, this is used as a pseudo linked list to find the most optimal route. Once this is finished the next node in the to visit queue is dequeued and added to the visited list, this becomes the current node, and the process starts over again. The loop ends when there are no more items in the to visit queue. The route is found by following the linked list created by the node data structure starting with the node that where the end found value is true.

**time complexity (worst case):** O(n\*v)

**time complexity (best case):** O(n\*v)

Where:

n = number of nodes in the matrix.

v = number of nodes in visited list; from 0 to n

**Explanation:**

The algorithm will traverse to every node in the matrix and in looking for new nodes will perform a linear search through every node in the visited list, whose length starts at 0 but ends up at n.

**Improvements:**

To remove the v value from the complexity, I could use a hash map to access a node in constant time. This was suggested to me by Mehemet after implementation, hence why I have not included it in the code.

I did think if I could use heuristics to optimise the algorithm by stopping it traversing to nodes that don’t lead in the direction of the end and make it more akin to something like A\* or Dijkstra’s but as the target node’s position is not known I could not see a way of making a prediction and know which paths to stop.

## 2.5 Incorporation of formative feedback

|  |  |  |  |
| --- | --- | --- | --- |
| Date and Time | Name of the Tutor | Feedback of the tutor | How the comment is addressed |
| 18/10/24 11:30 | Mehemet | Good work | N/A |
| 21/11/24 11:15 | Mehemet | Good work – consider the time/space trade off when using the visited data structure | N/A |

# 3. Project 3

## 3.1 **Data Structure selection**

Each node is a string and each edge for weighted graphs is an integer.

To represent the graphs, I used adjacency lists implemented as a dictionary in formats:

Unweighted graph: key: string nodeName value: list of strings nodeNames.

weighted graphs: key: string nodeName value: dictionary: key: strings nodeNames value: edge weights.

I decided to use dictionary so I could use the key index look ups so I wouldn’t have to do a linear search every time I needed to find a nodes adjacencies. Additionally, I wouldn’t have to format adjacent nodes I could just add them to the to visit queue when the program conducts the breadth first search. There is some data redundancy for the current solution as it stores both directions, however this means I can easily scale the algorithms to directed graphs.

# 3.2 Developed Algorithms

# Algorithm1: Influencers in Unweighted Network

A screenshot of a computer program

Description automatically generated

### Time Complexity of Algorithm1: Discuss about the time complexity of Algorithm1 that you have developed.

The findRoutes function is O(V+E)

The findRoute function is O(R)

The calcScore function is O(V\*R+ (V+E))

The findBestScore function is O(V(V\*R+ (V+E)))

Where:

V = number of vertices

E = number of edges

R = the number of vertices in that route

I think the time complexity shows the algorithm needs more refinement if it is to be used on a larger scale than the data set provided.

## 3.2.2 Algorithm2: Influencers in Weighted Network

A screenshot of a computer code

Description automatically generated

### Time Complexity of Algorithm2: Discuss about the time complexity of Algorithm2 that you have developed.

Same as algorithm1 *for ADL.*

# Implementation

## 3.3.1 Implementation details

I created two classes in C# for the respective problems and corresponding algorithms and used the two datasets provided to test each class to see if they worked. The unweighted graph worked fine and met the acceptance tests, the weighted one didn’t, prompting a redesign. The issue was that the algorithm was determining the shortest route by number of nodes visited instead of total weight. I redesigned the algorithm based on Dijkstra’s to find the route with the shortest weight.

## 3.3.2 Publishing your project

<https://github.com/Omagar1/ADS/tree/main/Project%203/Project3>

## 3.3.3 Evaluation

The Road Networks data set is significantly larger than the data set tested in the implementation and since currently I visit every node twice, the time my algorithm would take would be significantly larger. My program, however, would provide a 100% correct solution. To speed up my program I could use heuristics to determine the worst routes – reducing the number of nodes the program would traverse to speeding up the program; to best implement this solution it would take a significant redesign, and the resulting program would not be 100% accurate.

# Reflection

## 3.4.1 Video

Make a video presentation explaining the working of the developed program in C# for not more than 10 minutes.

## 3.4.2 Feedback

|  |  |  |  |
| --- | --- | --- | --- |
| Date and Time | Name of the Tutor | Feedback of the tutor | How the comment is addressed |
| 5/12/24 10:00 | Jaya Tangirala | Good work on code | N/A |
|  |  |  |  |

# 4. Project 4

## 4.1 Introduction to Bin Packing Problem

One dimensional bin packing is an NP-Hard problem meaning it cannot be verified quickly. It refers to a problem of packing varying sized items into fixed sized bins, with the aim of minimizing the number of bins required to pack all the items.

Worst-case scenario:

This is where every item is in its own bin, this would be forced to happen if every item is greater than the half the capacity of the bins.

A group of squares with numbers

Description automatically generated

*(Above) A visualisation of a worst-case solution for a simple version of bin packing*

Best-case scenario:

The minimum number of bins. This is difficult to find as it depends on the size of the bins and the items; larger bins and smaller items means is the best case, and smaller bins and larger items is the worst.

The website <https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/> uses the formula:

Min no. of bins >= Ceil ((Total Weight) / (Bin Capacity)

Where:

* Total weight is the sum of the size (or weight as used in the article) of the items
* Ceil is a function that rounds to the next greatest integer

Note the >=, this is because this formula assumes that the items fit perfectly into each bin, this is not always the case, hence why the problem is NP-Hard however this formula could be a good Jumping off point for an estimate of the number of bins needed.

A row of rectangular boxes with numbers

Description automatically generated

*(Above) A visualisation of a near optimum solution for a simple version of bin packing*

Complexity of the Problem

As mentioned before Bin Packing is NP-Hard, this is because it is an optimisation problem, not a decision problem. It’s complexity also increases exponentially with the number of items. To Illustrate this, I have used the previously mentioned formula to calculate the number of bins, so I plot the number of possible combinations compared to the size of the data set.

| Number of items to be Packed | Number of Possible combinations |
| --- | --- |
| 1 | 1 |
| 2 | 2 |
| 3 | 6 |
| 4 | 24 |
| 5 | 120 |
| 6 | 144 |
| 7 | 336 |
| 8 | 1920 |
| 9 | 12960 |
| 10 | 43200 |

Not all combinations are possible or optimal and this is using the optimal number of bins. This shows that large data sets would be very hard to calculate using a non-heuristic algorithm.

[*Spreadsheet link*](Project%204/Complexity%20Finder.xlsx)

*Other Citations:* <https://scialert.net/fulltext/?doi=jas.2013.919.923>

## 4.2 Random Mutation Hill Climbing

Random Mutation Hill Climbing (RMHC) is a greedy local search algorithm that uses a randomness to find solutions to a problem and a fitness function to determine how well the solution created solves the problem.

It works by generating an initial solution, evaluating its fitness, then making a small change and comparing it to the original solution’s fitness, if the fitness of the small change is higher it chooses the small change; if not it sticks with the initial solution, this is why it is called greedy – it only choses the solution with the better fitness. This is repeated until a solution with best fitness is found called an optima.

<https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/>

Pros of RMHC:

* Simplicity and ease of implementation
* Versatility
* Efficiency in finding local optima
* Customizability

There are problems with RMHC. Local optima is the most apparent; this is where the algorithm finds an optima, but this might not be the most optimal optimum hence why it’s called a local optimum, and since the algorithm is greedy it won’t move from the local optimum to find a better optimum.

A plateau is similar in that is caused by the greedy nature, but the surrounding solutions are of the same fitness as each other.

A diagram of a graph

Description automatically generated with medium confidence

*(Above) A visualisation of the local optima and plateau problems*

These problems can be overcome by a multitude of methods the two most common ones are:

* Random start RMHC – once local optima is found restart and see if the new local optima is better that the current local optima for x number of times
* Stochastic hill climbing is a non-greedy version of RMHC where it accepts a degree of worse fitness (determined by a tolerance value) in the hope of finding a better optima past the decline

## 4.3 Algorithm Design

### 4.3.1 Solution Representation

I plan on using an array of equal length to the data set; this array will contain the bin number I am assigning that index to. I am mapping the index of the solution to the data set – giving each item a bin.

A black and white picture of a number grid

Description automatically generated with medium confidence

### 4.3.2 Fitness Function Analysis

For all equations:

* n is number of bins
* b is a set of items in each bin
* c is the capacity of a bin
* is fitness

**Fitness equation 1:**

Mean average of the percentage of the bin used or negative integers for number of bins overflowing:

OR for all times >1: f –

I designed this fitness function to create a steady linear progression from worst case to best case in this way fitness should be steady.

**Fitness equation 2: from:** [**https://scialert.net/fulltext/?doi=jas.2013.919.923**](https://scialert.net/fulltext/?doi=jas.2013.919.923)

Mean average of the squared percentage of the bin used or negative integers for number of bins overflowing:

OR for all times >1: f --

A more exponential growth in the fitness value (FV). “This [fitness function] will avoid large plateaus in the search space around the best solutions” it provides a larger difference between the FVs of the best solutions though it gives less of a difference of FVs with worst solutions which may cause it to get stuck if the initial solution is bad.

**Fitness equation 3:**

A logarithmic approach accounting for the number of bins normalised to produce same range of values as previous equations:

OR for all times >1: f –

Does the opposite of equation 2, aimed at getting out of the worst-case scenario into a better scenario, is likely to plateau at higher values.

*data based on a small set of 3 bins*

A table of numbers and symbols

Description automatically generated

**Evaluation:**

Fitness 3 is more likely than the others to find a maxima in the in the mid ranges, so will likely find a solution but not the best solution because of this I believe fitness 1 will outperform it.

Fitness 2 is more likely to find a better solution once it passed the midpoint but would struggle to get out of a bad starting value. As the start is random it would likely need many restarts to find a good starting point to get the higher values.

Fitness 1 is constant and thus will have the same probability of improving from any starting solution which is good as the starting solution is random. Because of this I will choose fitness 1.

### 4.3.2 Small Change Strategy

I have a value called numChanges that is determined as the integer above 20% the size of the data set. For numChanges number of times I take a randomised index in the current solution and replace this value with a randomised number from 1 to the number of bins that I am using. I did consider swapping two randomised values but reasoned that this method has a lower potential ceiling in fitness as it only used bins generated by the initial solution; and if the number of bins was increased, as my algorithm does, then the swapping method couldn’t take advantage of that.

A diagram of a number

Description automatically generated with medium confidence

### 4.3.4 Data Structure

I will use arrays to store the data set and solution (of type double and integer respectively) as their sizes can be fixed and arrays allow for quick random access. I will use their indexes to map the solution to the data set.

So, the user can have a less abstract view on the way the bins are packed I will use as dictionary with the bin number as the key and the total of the items packed in each bin as the value to display the bins.

The output of the program will be a custom structure containing: the fitness of the solution as a double, the number of iterations the program took as an integer, the solution, and the bins both in the data structures previously mentioned.

### 4.3.5 Algorithm Pseudocode

|  |  |
| --- | --- |
| Line N.O. | Code |
| 1 | Input dataset |
| 2 | Input binCapasity |
| 3 | Let numChanges = length(dataset) \* 0.2 |
| 4 | Let numBins = sum(dataset)/ binCapasity rounding up |
| 5 | Let S be an array of random intigers |
| 6 | Let F be S’s fitness |
| 7 | While(noChangeCount > tolerance): |
| 8 | Let newS be an s with numChanges number of random changes |
| 9 | Let newF be newS’s Fitness |
| 10 | If newF > F: |
| 11 | Let S = new S and F = newF |
| 12 | Set noChangeCount to 0 |
| 13 | Else If fitness > 0 and noChangeCount >= tolerance -1 |
| 14 | Increment numBins |
| 15 | Set noChangeCount to 0 |
| 16 | Else |
| 17 | Increment noChangeCount |
| 18 | End If |
| 19 | End While |
| 20 | OUT S and F |

### 4.3.6 Experiment Strategy

My experiment strategy works by using a tolerance value that is the number of times the program will run until it outputs a result. However, if there is an overflow, represented by a negative integer value of fitness the program will increase the number of bins available by one and reset the no change count to zero. I do this to encourage the finding of valid solutions that are produced as I expect the more iterations the better the solution will be. I generally set the tolerance value to 100,000 though the optimal number would need to be determined, this would be based on the convergence value of the algorithm.

I present the results in a custom data structure called a BinPackingSolution which contains: the best solution found, the fitness of the solution, the iterations and the bins as they are filled by the solution. This can be written to a file.

## 4.4 Results

*Find attached* [*Spreadsheet*](Project%204/results.xlsx) *as data too large to present here*

This graph ended up a lot more discrete than I expected: I was expecting more of a bell curve from 0 fitness value (FV) to near 1. In fact, I got only 4 different FVs: -1, 0.776307692, 0.841 and 0.917454545 see Pi chart to see numbers of each FV.

I initially believed this discreet nature is caused by my experiment strategy of how I increment the number of bins if there in an overflow and reset the counter. My expectation was that if I did not do that, the results towards the higher number of iterations would be in the negatives. This is supported by the outliers of FV -1 which all have high iteration numbers. However, when I plotted the number of iterations against the number of bins it showed no correlation:

As such I don’t know why I have such discrete number of FVs.

The first graph shows the number of iterations needed to get to a high solution (known as the convergence value) is around 300,000.

## 4.5 Testing

### 4.5.1 Unit testing:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data Set: |  |  |  |  |  |
| index: | 0 | 1 | 2 | 3 | 4 |
| Data: | 4 | 2 | 3 | 1 | 2 |
| bin capacity: | 5 |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Function | Test Solution | Expected Result | Actual result | Passed | fix if necessary | Note index |
| GetBins |  |  |  |  |  |  |
|  | 1,2,2,1,3 | 1:5,2:5,3:2 | 1:5,2:5,3:2 | TRUE | N/A |  |
|  | 2,1,3,2,3 | 1:2,2:5,3:5 | 2:5,1:2,3:5 | TRUE | N/A |  |
|  | 1,2,3,4,5 | 1:4,2:2,3:3,4:1,5:2 | 1:4,2:2,3:3,4:1,5:2 | TRUE | N/A |  |
|  | 1,1,1,1,1 | 1:12 | 1:12 | TRUE | N/A |  |
| CalcFitness |  |  |  |  |  |  |
|  | 1,2,2,1,3 | 0.8 | 0.8 | TRUE | N/A |  |
|  | 2,1,3,2,3 | 0.8 | 0.8 | TRUE | N/A |  |
|  | 1,2,3,4,5 | 0.48 | 0.48 | TRUE | N/A |  |
|  | 1,1,1,1,1 | -1 | -1 | TRUE | N/A | 1 |
| SmallChange |  |  |  |  |  | 2 |
|  | 1,2,2,1,3 | N/A | 1,2,2,1,3 | TRUE | N/A |  |
|  | 2,1,3,2,3 | N/A | 2,1,2,2,3 | TRUE | N/A |  |
|  | 1,2,3,4,5 | N/A | 1,2,3,4,5 | TRUE | N/A |  |
|  | 1,1,1,1,1 | N/A | 2,1,1,1,1 | TRUE | N/A |  |

**Notes:**

1. *– 1 is an overflow as 12 > 5*
2. *No expected solution as small change is random; pass defined as true if the sum off all elements of result is within the range of the original solutions sum +- number change \* number of bins*
3. *There is no erroneous data as all data is generated by the program and not by a user thus no need to test a case that wouldn’t come up.*

### 4.5.1 System testing:

*Find attached* [*Spreadsheet*](Project%204/results.xlsx) *as data too large to present here*

## 4.6 Software and its Presentation, including testing (and video link)

*All you need is to attach a link for the video recording. Do not forget to ensure the video recording is accessible by us.* **If we cannot access your video recording, you will get zero marks for that section. Therefore, please ensure you test your video link by asking someone else to access your video recording.**

## 4.7 Incorporation of formative feedback

|  |  |  |  |
| --- | --- | --- | --- |
| Date and Time | Name of the Tutor | Feedback of the tutor | How the comment is addressed |
| 15/11/24 | Ziarul | Discuss why the problem is so complex | Added complexity sub heading in 4.1 |
| 21/11/24 | Ziarul | Good fitness function; add file reading | Added file reading |