Digital Systems Project

Particle Swarm Optimization to be embedded with a reinforcement learning algorithm to solve optimization problems.

Tom South

18011748

University of the West of England, Bristol

UFCFXK-30-3

# Abstract

This report proposes a swarm intelligence algorithm that will work with multiple operators which will also include a reinforcement learning algorithm and the algorithm will be able to solve binary problems such as the 0-1 multidimensional knapsack problem. “Swarm intelligence concerns the collective, emerging behavior of multiple, interacting agents that follow some simple rules.” Swarm intelligence acts like collective intelligence where agents show organization in their behavior. The swarm intelligence algorithm will show the behavior of the swarm moving towards an optimal solution of test data. (Hassanien and Emary, 2016)

A program will be designed, implemented, and tested using different information collected during research, using ideas found in literature read and even taking ideas of areas not been fully identified. The purpose of this program is to use a reinforcement learning algorithm to teach the swarm intelligence algorithms operators, such as mutation, to solve binary problems.

# Acknowledgements

I would like to thank my supervisor, Dr. Mehmet Aydin for reviewing my work and providing valuable feedback throughout all the stages conducted throughout this project. Mehmet discussed all my ideas and brainstormed further areas into which I could pursue and investigate that had not been discussed before in my project. Mehmet has further continued his support and guidance for throughout this project and overall, in the academic year.

# Table of Contents

[Abstract 2](#_Toc68547881)

[Acknowledgements 3](#_Toc68547882)

[Table of Contents 4](#_Toc68547883)

[Table of Figures 6](#_Toc68547884)

[Table of Tables 8](#_Toc68547885)

[1. Introduction 9](#_Toc68547886)

[2. Literature Review 9](#_Toc68547887)

[2.1 Previous Projects 14](#_Toc68547888)

[2.2 Identified Gaps 16](#_Toc68547889)

[2.3 Project Scope 16](#_Toc68547890)

[3. Requirements 17](#_Toc68547891)

[4. Methodology 18](#_Toc68547892)

[4.1 Planning 19](#_Toc68547893)

[4.1.1 Sprint 1 19](#_Toc68547894)

[4.1.2 Sprint 2 20](#_Toc68547895)

[4.1.3 Sprint 3 20](#_Toc68547896)

[4.2 User Story 20](#_Toc68547897)

[5. Design 21](#_Toc68547898)

[5.1 Class Diagram 22](#_Toc68547899)

[5.2 Activity Diagram 23](#_Toc68547900)

[5.3 Pseudocode 24](#_Toc68547901)

[5.4 Interface 28](#_Toc68547902)

[6. Implementation 28](#_Toc68547903)

[6.1 Sprint 1 (Particle Swarm Optimization) 29](#_Toc68547904)

[6.2 Sprint 2 (Multiple Mutation and Adaptive Selection) 31](#_Toc68547905)

[6.3 Sprint 3 (Q-Learning algorithm) 33](#_Toc68547906)

[7. Results and Findings 36](#_Toc68547907)

[8. Project Evaluation 39](#_Toc68547908)

[8.1 Limitations 39](#_Toc68547909)

[8.2 Improvements 39](#_Toc68547910)

[8.3 Reflection 40](#_Toc68547911)

[8.4 Feedback 41](#_Toc68547912)

[9. Conclusion 42](#_Toc68547913)

[10. References 42](#_Toc68547914)

[10.1 Literature 42](#_Toc68547915)

[10.2 Software and Libraries 44](#_Toc68547916)

[11. Appendices 45](#_Toc68547917)

[11.1 Appendices 1: Data Sets 45](#_Toc68547918)

[11.2 Appendices 2: Extraction of Q-Table from Implementation 46](#_Toc68547919)

# Table of Figures

[Figure 1: Reinforcement Learning Flow. 10](#_Toc68547941)

[Figure 2: Q-Learning Bellman Equation 11](#_Toc68547942)

[Figure 3: Multidimensional Knapsack Problem (MKP) 12](#_Toc68547943)

[Figure 4: Gantt Chart 19](#_Toc68547944)

[Figure 5: Class Diagram 23](#_Toc68547945)

[Figure 6: Activity Diagram 24](#_Toc68547946)

[Figure 7: Initialization Pseudocode 25](#_Toc68547947)

[Figure 8: PSO Pseudocode 26](#_Toc68547948)

[Figure 9: Adaptive Selection Operator using Probability Matching Pseudocode 27](#_Toc68547949)

[Figure 10: Q-Learning Pseudocode 28](#_Toc68547950)

[Figure 11: Example of data used during Implementation (w/ Description) 29](#_Toc68547951)

[Figure 12: Particle altering algorithm. 30](#_Toc68547952)

[Figure 13: PSO Single Mutation algorithm. 31](#_Toc68547953)

[Figure 14: Multiple Mutation algorithm. 31](#_Toc68547954)

[Figure 15: Adaptive Selection algorithm. 32](#_Toc68547955)

[Figure 16: Adaptive Mutation Selection Operator 33](#_Toc68547956)

[Figure 17: Q-Learning algorithm. 35](#_Toc68547957)

[Figure 18: Comparing Mean Fitness using Data Set 5 37](#_Toc68547958)

[Figure 19: Comparing Best Fitness using Data Set 5 37](#_Toc68547959)

[Figure 20: Comparing Mean Fitness using Data Set 6 38](#_Toc68547960)

[Figure 21: Comparing Best Fitness using Data Set 6 38](#_Toc68547961)

# Table of Tables

[Table 1: Requirements 17](#_Toc68102364)

[Table 2: Data Sets 38](#_Toc68102365)

[Table 3: Extraction of Q-Table 39](#_Toc68102366)

# Introduction

Optimization means finding the best solution among many feasible solutions that are available to us. You base a function off its performance, and the aim is for the function to be maximized/minimized (Arora, 2015). Swarm intelligence-based algorithms are from an area called bio-inspired algorithms, e.g. genetic algorithms are also from this area, as they are natured inspired, the particle swarm optimization I will be using is natured inspired, also are others such as the ant colony optimization (Hassanien and Emary, 2016). Reinforcement learning agents are among the most advanced and very capable of demonstrating high level of intelligence and rational behavior (Sewak, 2019). Reinforcement learning interacts with its environment and due to the environment it is in, can change many different states, it can use a point system as a reward system, based on how it changes the environment it is in and the objective it wants to complete. Sewak (2019) states that reinforcement learning led to the development of many advanced AI agents to perform very complex tasks, sometimes even challenging human performance at specific tasks.

For this project, research will be conducted on how swarm intelligence, particularly particle swarm optimization can be used to solve such binary problems, where the 0-1 multidimensional knapsack problem will be intensively researched as the main problem to be solved. Reinforcement learning will be researched and developed, as reinforcement learning rewards agent is dependent on its actions therefore creating a learning process over time. The aim is for the reinforcement learning algorithm to teach the optimization algorithm how to solve this problem. (Ravichandran, 2018).

One of the problems being tested is the 0-1 multidimensional knapsack problem, this problem arises in different scenarios, Boyer, Elkihel and El Baz (2009) states that these problems are found in cargo loading, cutting stock problems and processor allocation. This problem will be intensively researched and will be the benchmark for testing the algorithm.

# Literature Review

Arsham (2005) states that “A mathematical optimization model consists of an objective function and a set of constraints in the form of a system of equations or inequalities”. Which defines the problems I plan to solve during this project. Optimization problems are available everywhere, but they all have different characteristics and most require a specific technique to find solutions. They are classed based on the mathematical characteristics of the objective function, the constraints, and the controllable decision variables. (Arsham, 2005)

Nandy and Biswas (2018) state how reinforcement learning starts with an intelligence program, known as agents, and when they react with environments, there are rewards and punishments, environments can be known or unknown to the agents. These agents take specific actions to move to continuous states so that they reach their goal by maximizing the rewards given. Figure 1 shows the flow of reinforcement learning as stated by Nandy and Biswas (2018). The relevance of the flow of reinforcement learning to the project is to understand how the agent in this case particles, react with the environment to alter their data, which then leads to a change of fitness. Reinforcement learning can teach the algorithm to not alter significant pieces of data that would reward a negative value. Therefore, allowing for the progress of learning to be significantly faster if this were to happen over many generations.

Diagram

Description automatically generated

Figure 1: Reinforcement Learning Flow.

Sewak (2019) details how the reward is a function of both the action and the state, not just the action by itself, therefore the same action could (and ideally should) receive a different reward under different states, which under real-life terminology explains how using different actions in different aspects and areas of life, give different rewards and consequences. This details how impressive the approach of reinforcement learning is compared to other learning approaches, as it considers the environment in which actions are taken. Sewak (2019) and Nandy and Biswas (2018) detail the base of reinforcement learning, Nandy and Biswas show a very important flow of how the agent has a relationship with the environment and the factors it goes through. There are many different areas in reinforcement learning such as Q-Learning which are areas that could be studied further into which would allow this algorithm to enhance its learning.

Violante (2019) explains the basics of Q-Learning and its use in artificial intelligence. “Q-Learning seeks to find the best action to take given the current state” (Violante, 2019) Q-Learning seeks to maximise the total reward possible, therefore it being very useful in the use of optimization, where the aim is to optimize the problem as much as possible. The agent reacts to the environment it is and updates a table which is detailed with the state and action pairs to certain events. The agent can view all possible actions for a given state and choose based on the values of those actions, known as exploiting, or it can work randomly, known as exploring. Violante (2019) explains 3 basic steps on how a Q-Learning algorithm updates its (state, action) matrix table:

* Agent starts in a state (s1) takes an action (a1) and receives a reward (r1).
* Agent selects action by referencing Q-table with highest value (max) OR by random (epsilon, ε).
* Update q-values.

Paul (2019) continues the research of Q-Learning by introducing the Bellman Equation, which calculates the value of being in a particular state and considers all actions and chooses the action that would return the maximum value.

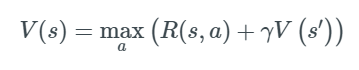


Figure 2: Q-Learning Bellman Equation

Paul (2019) explains the equation where:

* s = a particular state
* a = action
* s’ = state to which the agent goes to from s
* 𝜸 = discount factor
* R(s,a) = a reward function, taking inputs of a state (s) and action(a) and outputs a reward value e.g. 0 or 1.
* V(s) = value of being in a state

Comparing the research of Paul and Violante, they both explain how Q-Learning works with Violante showing more interest into the implementation of an algorithm via Python. Paul detailed more into the mathematics behind the equation, showing more diagrams on how an agent reacts with an environment, the movement from the calculations of the Bellman equation and so forth. Using both pieces of information give great understanding to the mathematics and what is happening in each state and to see how this equation is implemented with coding examples. Q-Learning will also allow to get the best reward possible per state and action, using Q-Learning should allow the algorithm to get significantly better fitness’s over generations.

Haddar et al. (2016) state that the particle swarm optimization algorithm is a heuristic global optimization method originally introduced by Kennedy and Eberhart in 1995 and that the algorithm exploits the concept that knowledge is needed for the search of an optimal solution and can be based on observed social behavior. Giftson Samuel and Christober Asir Rajan (2015) follow the previous statement by explaining how it is inspired from the collective behavior stated above in swarms of social insects. It explains how a swarm is made up of many particles and each particle represents an individual. Each particle is dedicated their own position and flight velocity, which during the optimization process becomes adjusted.

They continue to explain how the initial particles and velocities get initialized through random selection, then the velocities get updated by updating the fitness of each particle in the swarm, then the position of each particle is updated. Using this knowledge of particle swam optimization, previous works of using PSO and reinforcement learning in binary problems can be researched to find areas where gaps occur. This paper indicates the underlying concept of particle swarm optimization and how to use it in the project, it introduces the concept of putting binary data within particles and fitness’s, so that data is always within the particle as it moves to try to get to an optimal value over generations.

Punchinger et al. (2010) researched the structure and algorithms of the MKP and as shown in Figure 3 that the MKP can be defined as:

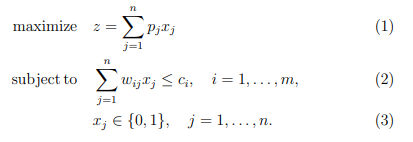


Figure 3: Multidimensional Knapsack Problem (MKP)

Puchinger et al. (2010) follow up this formula by explaining “A set of *n* items with profits *pj* > 0 and *m* resources with capacities *ci* > 0 are given. Each item j consumes an amount *wij* ≥ 0 from each resource *i*. The 0-1 decision variables *xj* indicate with items are selected.” The goal of the problem is to choose a subset of items with maximum total profit. But selected items must not exceed some resource capacities, which are expressed by the knapsack constraints in (2) of Figure 3. This can be linked back to Arsham (2005) where the objective function is the optimal value of the solution, and the set of parameters include the capacities of each knapsack, the size of each object and the weights of each object being added. The constraints are the relations between the parameters explained and the decision variables (Arsham, 2005), this can be shown in the 0-1 MKP as the decision variables are operators that I will use during my particle swarm optimization to get certain values to alter my objective function, which would be the total fitness of a specific particle solution. I can link this paper also to Hadder et al. (2016) and Giftson Samuel and Christober Asir Rajan (2015) where combining all the information from these papers, you can get a good contrast of how within particles, data will be stored and how it can be stored and accessed and then through the particle swarm optimization be manipulated to get its best solutions. Combining all the research so far allowed me to research into topics that further optimized the solutions, such as operators to add on to the existing PSO and RL algorithm.

Adaptive Selection is a selection operator which uses probability matching to find ways to select new solutions. Sharma et al. (2018) defines Adaptive Operator Selection as a “framework that dynamically selects an operator at run-time from a finite set of choices”. “The quality of each operator is calculated as the weighted sum of a reward value, which measures the impact of the most recent application of the operator on solution fitness, and its historical quality” (Sharma et al. 2018), this predicts the quality of an operator based on a method and is inspired by the Bellman equation from reinforcement learning shown in Figure 3 (Sharma et al. 2018).

“The Operator Selector estimates the quality q*i,t+1* of each operator *i*, based on the reward assigned to it at iteration t, and chooses one operator to use in iteration *t*+1 among *K* operators according to its quality” (Sharma et al. 2018). The idea proposed is to use multiple mutations to find different solutions, the Operator Selector described above will be able to choose one of those mutations based on the credit score that mutation operator has, which will be incremented every time it is used. Throughout generations or when max generations have been reached or even when an optimal solution has been found, it will be possible to see how many times each mutation operator has been selected. I believe this paper to be a great area to add to the existing PSO algorithm, this research introduces the intention to select a mutation operator based off its past behavior and through probability matching this should show an increase in performance in getting to an optimal solution.

This operator will be one of the main operators of the program, with multiple mutation as they connect very well with each other, using the probability matching that adaptive selection allows to select certain mutations based off the behavior of that mutation model in past generations. This operator can then be compared to the PSO with the reinforcement learning algorithm to compare performance between algorithms in reaching optimal solutions within multiple ranging data sets.

## 2.1 Previous Projects

Reinforcement learning can be used in many different aspects, Ciaburro (2018) details different examples where he has used reinforcement learning in different use cases with different algorithms. The main one researched is the Balancing Cart Pole which uses OpenAI Gym framework, which is used on Python. It is also a traditional problem and a well-known problem. Ciaburro (2018) explains how the objective is to stabilize the pole, without it falling to the ground, the agent will choose to move the cart left or right, and it will receive a reward every time the pole is balanced. The problem is a simple binary classification problem as it contains four inputs (cart position, cart velocity, pole angle, pole velocity at tip) and one single binary output.

Arin and Rabadi (2017) explain how “machine learning approaches have been successfully applied in optimization problems whose output is a sequence of actions, or an optimum policy”. Their approach is to test Q-Learning on the 0-1 multidimensional knapsack problem to compare it against other techniques, such as the estimation of distribution algorithms. They combine Q-Learning with Meta-RaPS which is a “generic, high-level strategy that modifies deterministic greedy algorithms by randomizing priority rules” (Arin and Rabadi, 2017). The use of using Q-Learning with other techniques gives the understanding of the connection between reinforcement learning and the other techniques and gives knowledge of connecting the PSO and the Q-Learning algorithm together using a Q-Table format.

Azad, Rocha and Fernandes (2014) created an artificial fish swarm algorithm for the 0-1 MKP where the artificial fish uses the entity of a real fish, where the school is the population of fish, therefore the environment that the fish moves in, searching for the best solution, is the search space of the MKP problem. The fish in the swarm go through a chasing and swarming behavior, which they show through an algorithm to compute the central point which is the point closest to all other points, the swarm then “searches” and behaves randomly based on the “visual scope” They have a penalty function method to handle the constraints, very similar to what you see from the research in reinforcement learning, where it is penalized/rewarded dependent on the state of the environment. The program then goes through improving the feasible points given, which then they select a new population. Researching this journal containing this project was very interesting as it had a similar sense to what was intended of this project, just with particle swarm optimization, even though how it is handled is different, the overall basic outline is shown on how to handle this problem using swarm algorithms.

An implementation of Particle Swarm Optimization on the 0-1 MKP written in Java was found, by author TMats on GitHub created in 2017, reading through the code and understanding what each function, class was doing you could see how PSO works with this problem, this enables comparisons against pseudocode created to make sure the implementation of the algorithms is correct. Having a clear understanding of the problem ahead after reading this piece of code was helpful in going forward with planning and implementation as it gave great insight to what was expected to be implemented.

## 2.2 Identified Gaps

Many projects that were researched such as TMats (2017) and Azad, Rocha and Fernandes (2014) focus around the area of using a simple swarm algorithm to solve these binary optimization problems. Therefore, an area could be found where this problem has not been optimized, such as adding multiple mutation with the adaptive operator selector and the addition of Q-Learning. Q-Learning has been used in many optimization problems, such as the 0-1 MKP as detailed by Arin and Rabadi, but there seems to be the gap of applying reinforcement learning with a swarm intelligence algorithm to optimize these problems. This is the objective of this project.

## 2.3 Project Scope

The research into swarm intelligence algorithms is very interesting and very broad, the options from Ant Colony, to using an artificial fish swarm like Azad, Rocha and Fernandes (2014), this project expands on existing findings in simple particle swarm optimization by adding multiple different operators and an adaptive mutation selection and also a reinforcement learning algorithm. These algorithms can then all be compared in performance to gather results and findings.

The gap between this project and other projects researched which contain optimization with binary problems such as the 0-1 MKP is the choice of operators, using adaptive search to choose a mutation out of a choice with bias if it is chosen and further using that adaptiveness to rechoose that mutation in further generations. The use of a Q-Learning algorithm with my particle swarm optimization introduces a reinforcement learning scheme and from research, a reference was unable to be found that implements particle swarm optimization with reinforcement learning, particularly Q-Learning and using the other operators on the 0-1 MKP problem.

A system will be design and developed taking findings from research and the previous implementations mentioned. The aim of this implementation is to take a simple generic swarm intelligence algorithm and expand on how it can be implemented using other artificial intelligence algorithms.

# Requirements

For the requirements for this project, areas were investigated that were discussed in the literature review, especially highlighting research on the 0-1 MKP problem and the basics of particle swarm optimization and Q-Learning. Research from Puchinger especially gave ideas for non-functional requirements in relation to how data should be structured and so forth. Research from Haddar, Giftson Samuel and Christober Asir Rajan gave ideas on how to structure the functional requirements dependent on how data should be output per generation and once optimum solutions have been found and so forth. For the non-functional requirements, the efficiency, usability, and performance of each algorithm had to be considered and highlighted areas of discussion on how these different areas can be shown and detailed specifically.

There are 7 functional requirements and 5 non-functional requirements listed below which are the outline for what I expect my program to do, and how I expect data to be outputted/shown. These requirements either use “must”, “should” and “could” to prioritize what requirements are necessary and what requirements are not but would be a good addition to the program. My requirements use the MoSCoW method as a form of showing prioritization.

|  |  |
| --- | --- |
| Requirement | Description |
| F1 | The program must be able to solve the 0-1 multidimensional knapsack problem. |
| F2 | The program must have a working reinforcement learning algorithm that rewards/penalizes the agent due to the state of the environment it is in. |
| F3 | The program must have working operators within the particle swarm optimization, so it can optimize the problem and get a solution. |
| F4 | The program must be able to show that optimization has happened and that it is solving the 0-1 multidimensional knapsack problem in stages. |
| F5 | The program should allow for the reinforcement learning algorithm to choose which operators should be run in the particle swarm algorithm. |
| F6 | The program should allow for the agent to work in all states of the environment allowing for a penalize/reward situation. |
| F7 | The program must have a working adaptive mutation selection operator where the performance is visible in terms of a graph created. |
| NF1 | The program must be efficient in how the algorithm optimizes the 0-1 MKP. |
| NF2 | The different versions of each program should be able to show the increase in performance of each algorithm over the course of the duration of the implementation. |
| NF3 | The data of each program should be structured equal to assure the algorithm has no problems optimizing the data. |
| NF4 | The program should be able to be reused using different data sets with each data set being accessible to all users. |
| NF5 | The algorithm could be used to show the performance of different areas such as selection, mutation, Q-Learning compared to other algorithms where these areas are also used. |

Table 1: Requirements

# Methodology

The Agile method has been decided while implementing this project, as this allows to receive feedback especially if problems arise during the implementation period, which have a high chance of occurring, especially during early implementation. This feedback can contribute to making changes in the correct areas and raise new ideas on how this project can be implemented.

Sprints have been introduced to spread out the implementation process which will be shown in the planning. Sprints were introduced due to experience in the field, it allows the creator to work to certain deadlines where feedback can be obtained during and after deadlines and these areas can be finalized and worked on whenever necessary. This can be an optimal way of designing and implementing the project. After each sprint, feedback will be obtained on the production so far and the aim is to make changes in accord to the feedback given and finally “clean up” any code that was implemented.

## 4.1 Planning

TeamGantt was used to create the Gantt chart due to the simplicity of the design, it also allows you to change it very easily as you go through the project, using percentages to show the completion so far of the project. This would be changed during the project to accommodate any changes occured in the order of my implementation / testing or any other action.

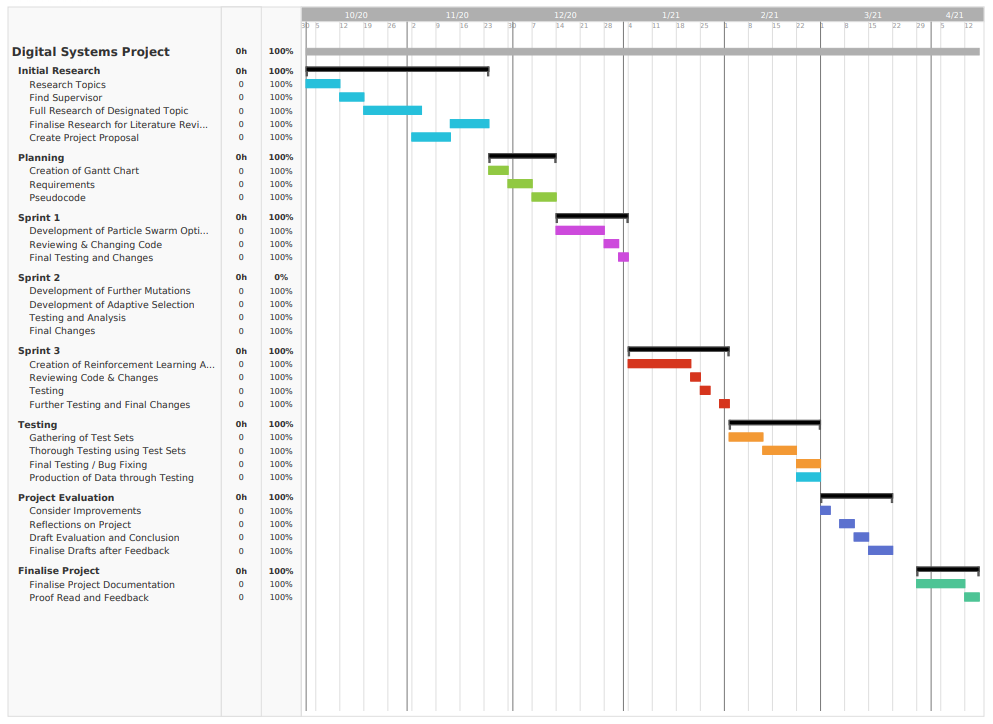


Figure 4: Gantt Chart

### Sprint 1

The first Sprint is designed to develop a particle swarm optimization (PSO) algorithm, which is the base of the program. During this period, a single mutation operator will be implemented to allow the global best to be optimized further and allow for particles to continuously follow towards an optimal solution. The successful implementation of a single mutation operator will indicate to move to the next sprint, to enhance this algorithm into a multiple mutation operator and finally an adaptive mutation selection operator. The PSO algorithm with single mutation will then be tested using multiple test data to make sure the algorithm can optimize problems and at the end of the sprint, there will be a time for feedback and changes.

### Sprint 2

The second sprint will continue using duplicate code from the first sprint, changing the mutation operator to allow for 3 different mutation operators, which will then be selected by a random number generator. This then can be enhanced by using this code to develop an adaptive mutation selection operator to choose the mutation based on its performance in past generations using probability matching on the percentages of how much each operator has been chosen. Once these two implementations have been fully developed, testing will occur using data sets to make sure it can optimize different types of solutions, graphs will also be created to see the credits throughout generations that have been given to each mutation operator, hopefully this will show that a successful working adaptive mutation selection operator has been developed. These two developments will conclude sprint 2 and move onto sprint 3, to implement the reinforcement learning algorithm into the particle swarm optimization algorithm which has been implemented in the past two sprints.

### Sprint 3

The third sprint is implementing the reinforcement learning selection scheme to reward/penalize the particle swarm optimization and choose the operators in the PSO to improve the performance. This process could be the lengthiest of the sprints as a brand-new algorithm needs to be created and the PSO might have to be changed to work with the RL algorithm, feedback will follow this sprint to return to make changes wherever necessary. The end of this final sprint will be designated to fixing errors within code if necessary and to add validation and other areas such as a menu to select a choice of data sets to optimize.

## User Story

A user story has been added to detail what would be expected from a user’s perspective using the program implemented, the user story has been created to detail in depth the functionality of the program, including the inputs and outputs and how you can connect with each algorithm to detail performance. Two different user stories have been created, one from the perspective of a user using the PSO algorithm and one from the perspective of the implementer and designer of the algorithm and how the knapsack problem is expected to be optimized.

|  |  |
| --- | --- |
| **Feature**: Particle Swarm Optimization | **Importance**: High |
| **Description**: As a user, I want to perform particle swarm optimization on different test sets so I can achieve highlighting the difference in performance of each algorithm implemented. | **Acceptance Criteria**: Given that the input data is structured correctly and read into the implementation, when the PSO has finished optimizing each particle at the end of each generation, then the best particle with the sizes of each knapsack should be outputted. |

Table 2: Particle Swarm Optimization User Story

|  |  |
| --- | --- |
| **Feature:** Knapsack problem | **Importance:** High |
| **Description:** As the implementer of this algorithm, I want to manipulate the data of each data set of the knapsack problem so that I can optimize the knapsack problem. | **Acceptance Criteria:** Given the particle swarm optimization every generation is successful, per generation when the best particle is returned, then the change in sizes of each knapsack problem should be within constraints and can clearly show that the sizes are staying within a range therefore showing for the optimum fitness it is trying to achieve the best sizes per knapsack possible. |

Table 3: Knapsack Problem User Story

# Design

Once the requirements had been set out for the project, I set out to design the implementation. The project has been designed so that if possible, all the requirements could be met successfully.

The design of the algorithm will be based of objects that are particles which will react with each other to find better solutions throughout my Particle Swarm Optimization. The structural model will be an array for the particles with each index containing binary data, fitness, position and velocity of each particle. These particles will cooperate with each other to find the global best and other particles will follow this particle to optimize its initial solution. There will be many changes in states of the objects, such as the running of data into variables from text files where the initial data will all be stored. The movement to different functions to calculate fitness’s, sizes of each knapsack etc. will show the change of state of objects in the algorithm and how different objects data is manipulated in different ways.

## 5.1 Class Diagram

Astah UML (2020) was used to design the class diagram before pseudocode was designed, a plan was introduced to divide the program into two steps, initializing the initial variables with solutions within constraints, and then the particle swarm optimization which optimizes the initial solutions to get to the optimal value. To do this, classes were created of MKP Data, Particle, Fitness and finally PSO. The class of MKP data represents all the data read by the text file, which will always be stored, this data consists of number of knapsacks, objects, sizes and weights of each object, capacity of each knapsack and finally the initial solution. The Particle class represents a single particle which will all be stored within an array, all the particles will then be stored in a particle array, which will be used in the PSO class. The particle class includes the position, velocity, fitness, particle best and velocity max/min. This also includes functions such as meeting capacity (which will be used in optimization to get valid initial solutions), updating fitness/velocity/position and particle best. There is also a class for the reinforcement learning, which will train the PSO to select the best operator to generate new solutions, this class describes a Q-Learning algorithm which gets the best reward from all possible states, this reward being the best fitness. There is also an adaptive selection class, which works off rewards from past generations using probability matching to optimize results. The figure below represents the class diagram.

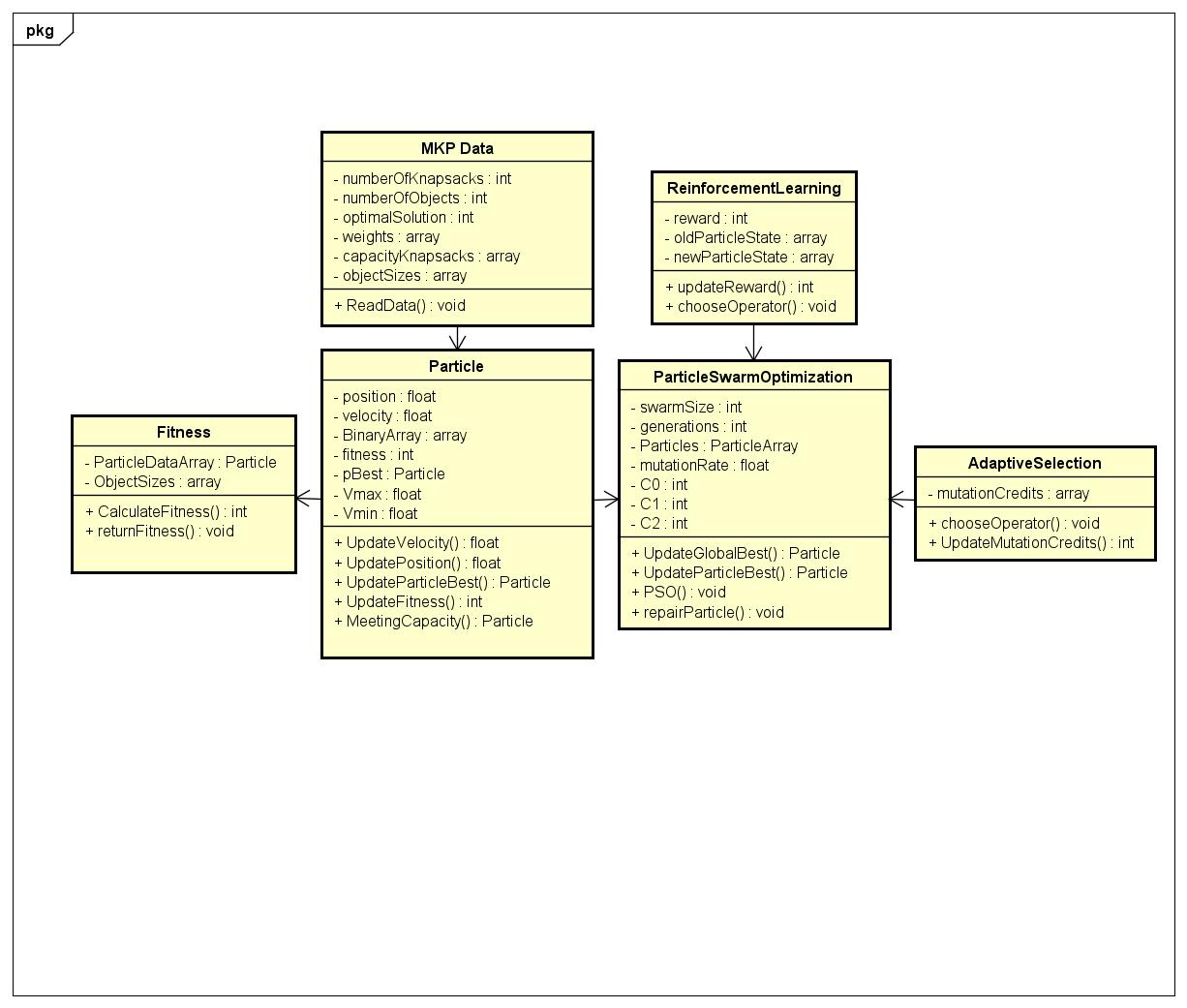


Figure 5: Class Diagram

## 5.2 Activity Diagram

For the activity diagram, I used Lucidchart (2018) to design the activities I wanted the particle swarm optimization with Q-Learning to go through, from initializing the variables, to optimizing every particle one-by-one throughout every generation. The activity diagram enables areas to look at when planning pseudocode and implementing this algorithm, such as the Q-Table entries and how to utilize the design of what will be entered so that the particle can take data from the Q-Table whenever necessary. Another idea created by the creation of this diagram is the use of a repair function, to repair the binary data to make sure the size of each knapsack is within the constraints to classify it as a valid solution.

Diagram, schematic

Description automatically generated

Figure 6: Activity Diagram

## 5.3 Pseudocode

From the diagrams created and research found from the past projects of the literature review, pseudocode could be created for the PSO created in steps as described in the class diagram of having initialization then optimization functions. To plan pseudocode for the initialization a way to create initial solutions had to be created. The particle data was always going to look like: [0,1,0,1,0,0,0][0,1,1,1,1,1][0,0,0,0,0,1] where the amount of arrays are the number of knapsacks and the length of a single array is the number of objects for each knapsack. 0 signals that an object would not be placed in that slot, and 1 indicates that an object would be placed in that slot, dependent on whether the size of the object was greater than 0. The class diagram shows that the particle will also have a position, velocity, and fitness. Therefore, each particle was designed to be shown as [0,1,…][1,0,…], fitness, position, velocity. Therefore, when all particles are stored in a whole array, each particle can be iterated through and the indexes would all be the same e.g. 0 for the binary data, 1 for fitness etc. Valid initial solutions would then be created by first initializing random 0,1’s into the binary array, then creating a function to randomly switch 0,1’s and calculating the sizes of each knapsack and if they equaled the capacities of their knapsacks, it was counted as a valid solution and a fitness could be calculated.After discussion with my supervisor and having draft pseudocode that was revised over time, final pseudocode for initialization was designed which was ready for implementation.

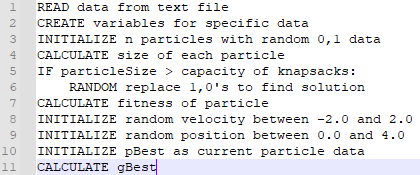


Figure 7: Initialization Pseudocode

PSO pseudocode could then be designed. From my research the PSO with the 0-1 MKP written in Java was very helpful to see how PSO was implemented and to get a clear understanding of what needs to be done speeded up this process very quickly. The plan was to design different drafts of how to implement the PSO, with discussion also with my supervisor, a solution was found quickly for the best method to implement the PSO. The PSO would iterate through generations, and a “newPopulation” array containing all the particle data which could be edited, to find new solutions and then once valid solutions have been optimized slightly, at the end of each generation, this “newPopulation” would update the initial population data. The program would iterate through each particle in each generation, changing data randomly that was similar to the global best, the program would then mutate this particle allowing it to optimize further, if it fit the constraints and the capacities it would be classed as a valid solution, updating velocity, position and then repeating for the next particle in the swarm. At the end of each generation, the best particle fitness would be outputted to show the progress of optimization over generations and the global best would be updated therefore next generation, particles are moving to the best solution produced so far. The initial population is then overwritten with the new population and the process repeats until the optimal solution or a maximum number of generations has been reached.

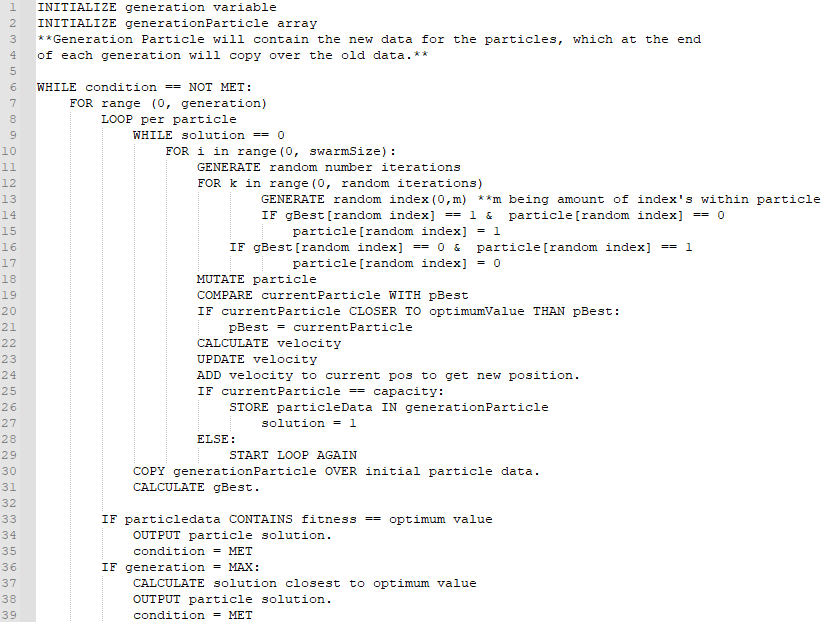


Figure 8: PSO Pseudocode

Once the basic design for my PSO with a single mutation operator had been designed, it was simple to change that single mutation into a multiple mutation operator, just creating 3 copies of the current data and mutating that data using the idea already designed.

An adaptive selection operator needed to be designed which would use probability matching of how much a mutation operator had been selected in past generations to affect the chance of it being chosen in the current generation. To do this, an idea was created that for the first iterations, to credit the mutations, it would choose a random number in the range of 0 to the total credits allocated so far. For example, if mutation operator 1 had been allocated 25 credits, mutation operator 2 been allocated 15 credits and mutation operator 3 been allocated 10 credits, this shows a 50:30:20 ratio. Through probability matching, if the random number is below or equal to 25, then mutation 1 is selected, if it lands on 26-40 therefore mutation 2 is selected and so forth. This allows for probability matching and the credits allocated in past generations to affect selection in further generations. This idea can be shown in Figure 9.

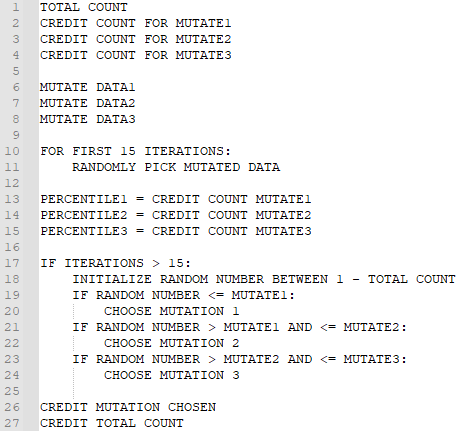


Figure 9: Adaptive Selection Operator using Probability Matching Pseudocode

To make comparisons to the adaptive selection with probability matching, a Q-Learning algorithm was designed to choose a mutation operator if its new state and reward is positive and holds a better fitness value than the original particle data. The idea of the algorithm is to calculate the fitness of the new mutations and put that against the particle fitness before it went through mutation. If the fitness is greater a positive reward is given, and a negative reward is given if it is less. The new q-table entry is then initialized and if it is in the Q-Table with a positive reward, that entry in the Q-Table is used. If the entry is in the table with a negative reward, no mutation is used and the original particle data before mutation is restored. If the entry is not within the Q-Table, it is entered into the table with its reward and that mutation is used as the particle data.

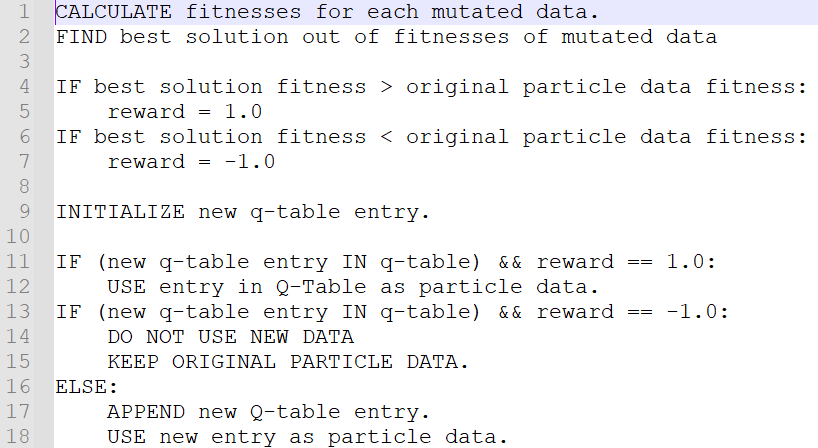


Figure 10: Q-Learning Pseudocode

## 5.4 Interface

For the interface, it will allow you to select a data set from 1-10 then it will display the optimum value for that data set and go through the optimization accordingly. Once the PSO begins, it will output generations of the PSO returning the best fitness in the swarm, this allows to see the initial best solution and the progress of optimization over time. Once the PSO has been complete, it will output if the optimum solution has been found and the binary data for that particle, with the knapsack sizes and fitness. If an optimum solution is not found, but the PSO has reached max generations, it will output the best particle found over all the generations, displaying all the relevant data for that particle.

# Implementation

During implementation, the sprints created during the planning stage will be used, this therefore split the implementation into three sub-implementations: the PSO with single mutation, implementing multiple mutation and adaptive mutation selection and finally implementing a Q-Learning version of the code.

## 6.1 Sprint 1 (Particle Swarm Optimization)

The project will be built using Python, some external libraries were needed such as numPy, random and matplotlib for generating random numbers, creating plots of graphs etc. These libraries can easily be installed through the command line, this directory allows to find the index of the global best particle, by finding the closest value to the optimal value. This proved very helpful.

To start the initialization, data sets were acquired from the Brunel OR-Library, created by J E Beasley (1990), this data was then changed so it was easier to read directly into variables, arrays during implementation.

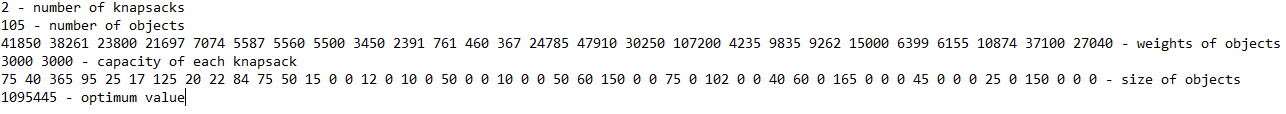


Figure 11: Example of data used during Implementation (w/ Description)

The weights, sizes and capacities are stored into arrays, therefore in the particle data, e.g. particle[1] will align to weights[1] when calculating fitness, which made things much simpler. Following the pseudocode shown in the planning stage, initializing data was very simple, where random 0,1’s had been initialized in an array for each particle. Functions were created to calculate the sizes of each knapsack which was simple, just iterating through the particle data and if the slot was 1, add the size of that index to the total.

To then create a valid solution, the sizes of the knapsacks created from the solutions needed to be equal or less to the capacity of each knapsack, to do this a function was created to randomly change values within the constraints to reach the capacity of each knapsack.

With the implementation of the PSO, I initialized variables for the generations and created a ‘newpopulation’ variable which would edit the population within a different variable and overwrite it at the end of every generation via the “deepcopy” function Python has within its copy library, this was in case any errors went on during the PSO which affected data. The creation of the PSO was split into 4 different pieces, one using a simple PSO algorithm with no operators, the second using the first algorithm but with a single mutation, which would be undertaken in this sprint. The third implementation would continue from the second piece and take place during sprint 2, having 3 different mutation operators and a random selection scheme and having a PSO with 3 different mutation operators and an adaptive selection operator. The final creation of the PSO was implementing the Q-Learning algorithm which would be undertaken during sprint 3.

The implementation of the simple PSO occurs by each particle ‘following’ the global best particle by altering its data slightly in relation to the global bests binary data, as there are n knapsacks with m objects, with >100 slots per knapsack, every generation each particle would alter between 10-15 pieces of data. The particle size would then be calculated and if it went over the capacity of any knapsack, it would enter a repair function to repair the solution to be within constraints.

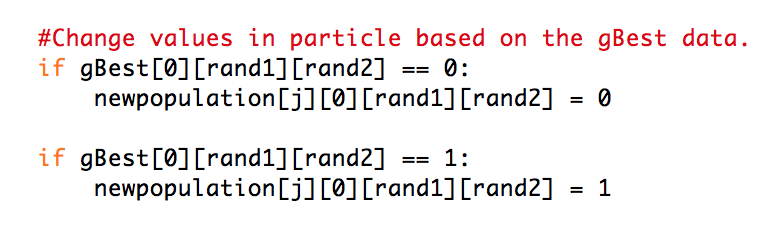


Figure 12: Particle altering algorithm.

Once it is classified as a valid solution, being within all constraints, the new velocity, position is calculated and the pBest is updated accordingly. This is then repeated for all particles every generation. At the end of every generation, the global best is calculated and updated, and its data is entered into an array to find the best solution in all generations if the maximum generations has been achieved. For the single mutation operator, A bit flip mutation operator was added which swapped binary data randomly from 0 to 1 or reverse. This would be added after the altering of data slightly compared to the global best and would just mutate the binary data to find better solutions during optimization.

By the end of this sprint, a fully functioning Particle Swarm Optimization algorithm was implemented, enhanced with a single mutation to let the global best of each generation optimize further, with the algorithm being tested on a default test set. The results of this algorithm compared to the other implementations will be compared during the testing of each algorithm.

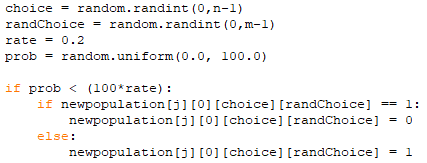


Figure 13: PSO Single Mutation algorithm.

## 6.2 Sprint 2 (Multiple Mutation and Adaptive Selection)

To change the single mutation to multiple mutation using different mutation operators two more mutation operators were added, interchanging and swap mutation. Therefore, all mutation operators would mutate 3 different sets of data all copied from the particle data following the simple PSO algorithm. After these mutations, a random number generator would select 1 of the 3 operators to be selected as the particles new data.

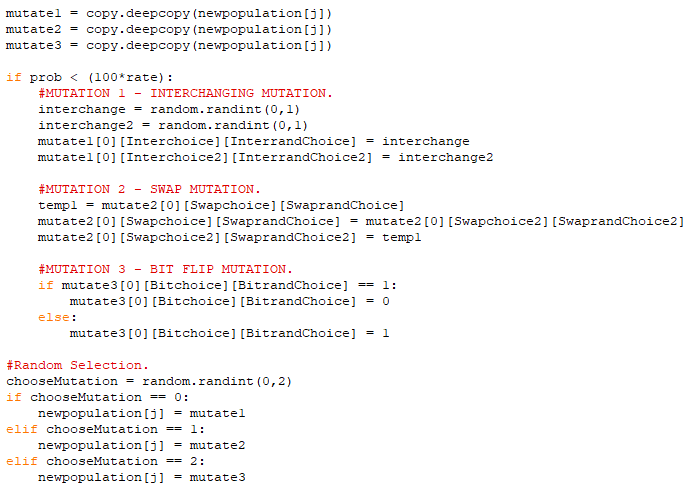


Figure 14: Multiple Mutation algorithm.

The random number generator was improved by adding an adaptive selection scheme which will choose a mutation operator based off its past experience in generations. For the first 15 iterations, each mutation was chosen randomly, and its reward was updated, each mutations reward would be incremented by 1 every time it was used. After the fifteen generations, dependent on how many times it was selected e.g., if there were a total reward of 20 and the first mutation had a reward total of 8, a random number generator would be selected and if it landed between 1-8, mutation 1 would be selected and so forth. Once max generations were reached, you could see through graphs and the array how many times each mutation operator was selected.

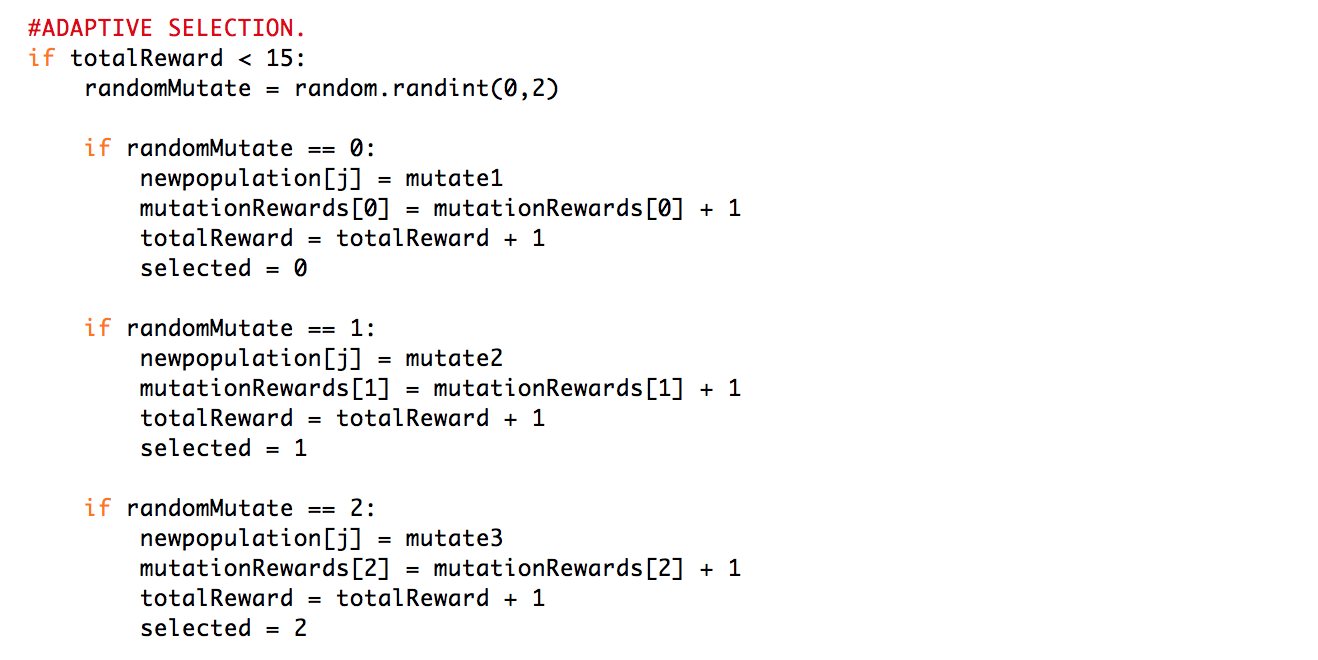




Figure 15: Adaptive Selection algorithm.

To test that the adaptive mutation selection operator worked during implementation, a function was created using matplotlib on Python, to plot a graph of the credits associated to each mutation operator throughout generations and the overall optimization of a data set. The graphs shown below in the figures show the credits awarded to each mutation operator and the graphs show the increase of the mutation operator used due to the credits increasing throughout iterations, this indicates an adaptive selection operator being successful during implementation.

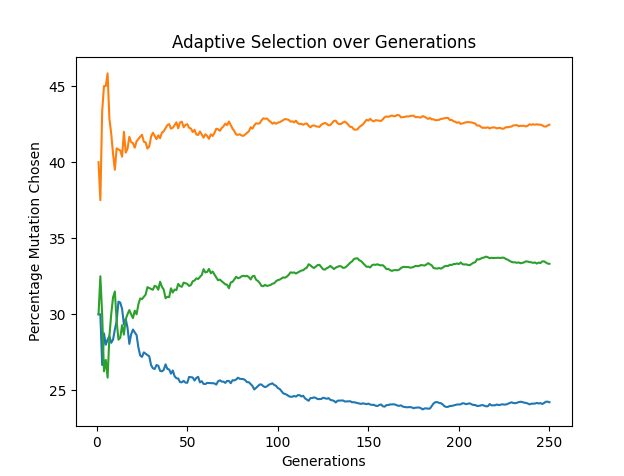
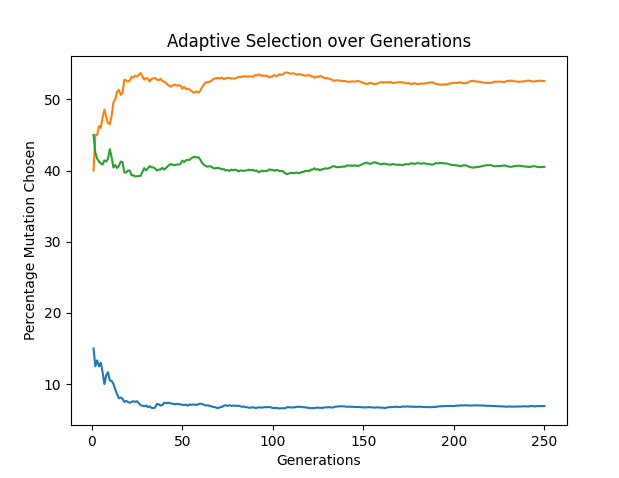


Figure 16: Adaptive Mutation Selection Operator

At the end of this sprint, the aims and objectives of having a PSO with multiple random mutation and adaptive selection using probability matching, the algorithms have been tested using a default data set from Brunel University, where an optimal solution can be found within 100 generations 50% of the time and almost 100% of the time within 200 generations. These algorithms will be then tested using further test sets where the performance in finding the optimal solution will be plotted and evaluated.

## 6.3 Sprint 3 (Q-Learning algorithm)

For the implementation of the third and final sprint, the implementation of the reinforcement selection scheme using Q-Learning, trying to implement the algorithm from the pseudocode produced from the design of this sprint.

To implement this algorithm, the Q-Learning table had to be initialized which would include the data from each Q-Table entry, each entry would include the reward, the new state (the mutated data particle), the previous state (the particle data entered before mutation). The three different mutation data go through there mutations and then the best fitness from the three mutations is selected. A reward is given dependent on whether the new fitness selected is better than the fitness of the original particle data before mutation. The new Q-Table entry is then created and if the Q Table entry is already an entry in the table with a positive reward, the data is used from that entry. If the entry is already within in the table with a negative reward, no mutation is therefore selected as it is not a positive reward and a better solution. Finally, if it is not an entry in the Q-Table, the new entry is added to the Q-Table, and the mutated data is used.

This algorithm was tested using the default data set and will be tested thoroughly during sprint 3, creating testing data which will be plotted against the other algorithms created in sprint 1 to compare performance. The finish of sprint 2 completed the final aim and objective of having an operating PSO with a Q-Learning algorithm as a reinforcement learning selection scheme. The algorithm implemented can be shown in the figure below and an extract of the Q-Table after a run using a default data set can be shown in the appendices.

To conclude Sprint 3, testing can now take place on all the implemented algorithms with 10 different test sets to collect data to use for the findings and to compare performance between algorithms. The data collected is the mean and standard deviation of each generation from 1-250 generations from 20 runs of each data set on each algorithm, therefore there will be 800 total runs across 4 algorithms. The data sets vary from different number of knapsacks and different numbers of objects, which the data sets will be shown in a table below. The maximum generations chosen of 250 is from testing using the default testing data used throughout implementation that certain algorithms can reach optimum value within those generations and some algorithms will not. This will show the performance of the algorithms when compared to each other, the results from the testing will lead to findings which will be able to create discussions due to the performance of these algorithms.

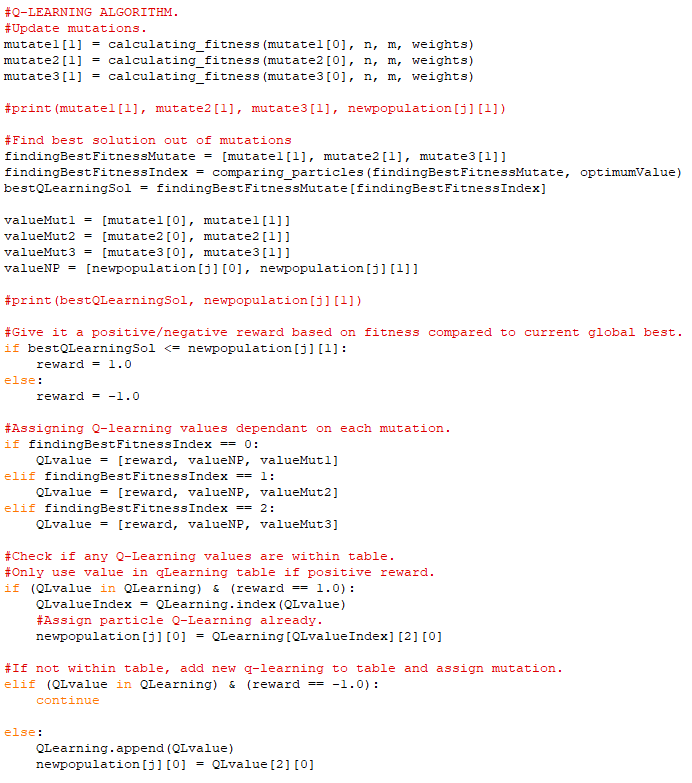


Figure 17: Q-Learning algorithm.

# Results and Findings

From the extensive testing, the results as shown from the testing results area in the appendices show how much performance the algorithm with Q-Learning has compared to the other algorithms, especially with larger data sets. As the knapsacks grew larger, with larger elements, overall, the performance of all algorithms decreased, the Q-Learning algorithm was the only algorithm that could find the optimum value and be within 1% of the optimum value on every run of the last 3 data sets. Compared to the adaptive mutation selection algorithm, the Q-Learning algorithm will always choose the best state for the particle to be in, therefore if the particle data before mutation occurs returned a better fitness than the fitness returned after mutation and selection, mutation therefore never occurs, and it continues throughout the process. However, with adaptive mutation selection it is based off the mutation selected through probability matching of its past generations, this means that after mutation there is a possibility that the fitness returned was worse than the fitness before mutation. This is believed to be a large factor in the separation in performance in these two algorithms.

The random mutation operator performed better than the adaptive mutation selection operator on selected data sets, this could be due to a range of reasons, one being that the probability matching adaptive mutation selection bases off its mutation operator selected by experience in past generations, therefore not always the best mutation is selected. With random mutation selection, there is a higher chance through the random selection that a better solution will be selected out of the three selections. Both these algorithms performed poor as data sets grew larger due to the lack of the best mutation (if it is possible) being selected, which is what the Q-Learning algorithm guarantees. The performance difference between the adaptive selection, random mutation selection and simple PSO algorithm was heavily shown when knapsacks grew larger in numbers and size, the algorithms could not reach within 10-15% of the optimum value due to the amount of data being manipulated at the time and the lack of possibility of a greater fitness always being found. All mutation rates were tested at 0.2, which during implementation was the best rate to be found, therefore all tests were equal between all algorithms.

Two different data sets were selected based on the results where there was a clear difference in performance of the algorithms, and the average and best fitness were taken per generation, therefore the performance difference can be shown through the graphs shown in the next figures, showing the much greater performance the Q-Learning algorithm has compared to the others.

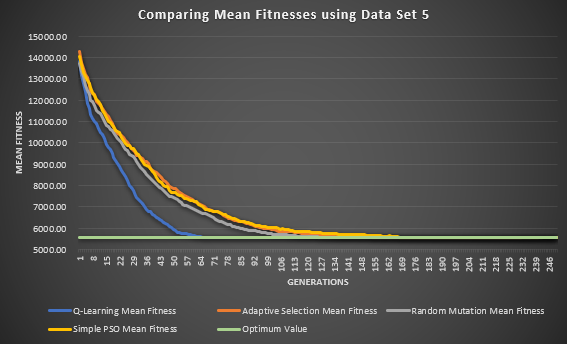


Figure 18: Comparing Mean Fitness using Data Set 5

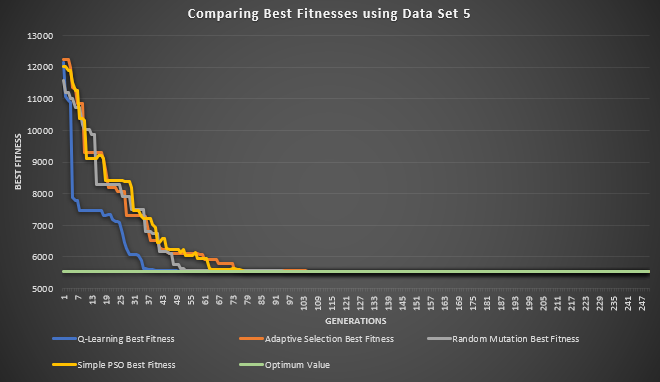


Figure 19: Comparing Best Fitness using Data Set 5

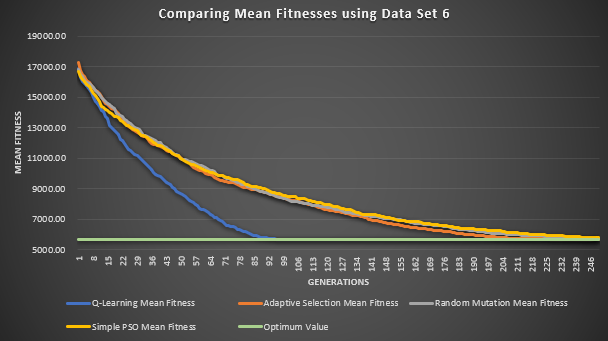


Figure 20: Comparing Mean Fitness using Data Set 6

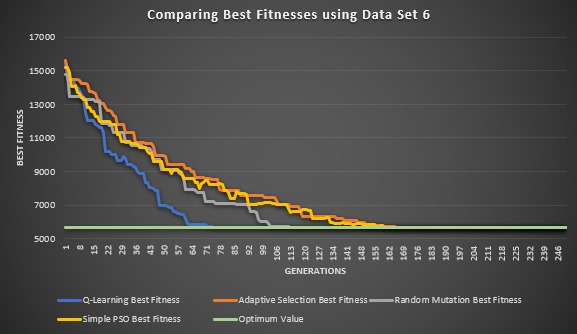


Figure 21: Comparing Best Fitness using Data Set 6

From the graphs created showing the results from two data sets, the performance of the Q-Learning algorithm is shown clearly, where it hits the optimum value in 48 less generations than any other algorithm in data set 5 and it hits the optimum in 118 less generations. The random mutation operator follows surprisingly in performance, even though the adaptive mutation selection was close until the latter stages of optimization. These results follow up on what was discussed earlier, stating that due to probability matching, having good performance in past generations does not always guarantee good performance in current generations.

In relation to the percentage of times the optimum value was “hit” by each algorithm, the Q-Learning algorithm had the best percentages of all the algorithms, followed by the adaptive mutation selection even though it had lower percentages than the random mutation operator in the mid-tier data sets, the adaptive mutation selection performed better in the latter data sets. The simplistic PSO algorithm including the single mutation operator performed as expected, working very well on smaller test sets but as the test sets grew within the generation range, it could not optimize the problem very well.

# Project Evaluation

## Limitations

The limitations for this program are simply that the data must be structured in a specific way that was given by the Brunel University in their data sets. This also includes the need for an optimum value for the program to stop once that specific value has been found. If there was not an optimum value within the data, the program would simply run technically similar to a minimization function. Once the program gets near to an optimum value it also tries to find the best sizes per knapsack, therefore if an optimum value were not given, it would not be able to do this.

In the current state of the program, if the outcome were to find the optimum value given a number of knapsacks, objects, weights and sizes without an optimum value, a simple function would have to be implemented to calculate an optimum value, which can be done simply using knapsack calculators and other research areas that are available.

## Improvements

In terms of improvements, we could enhance the learning of this particle swarm optimization algorithm by looking into the area of neural networks and Q-Learning by looking into deep Q-Learning which I believe could enhance the learning of this algorithm which would return even greater performance for optimizing larger data sets and would be a good comparison against the standard tabular Q-Learning algorithm implemented in this project.

To improve the current program, as spoken above the removal of an optimum solution from the inputted data, a function to calculate an optimum value given an input of number of knapsacks and number of objects followed by an array of weights and sizes could be implemented. This would allow for a full optimization of any random 0-1 multidimensional knapsack problem given a structured input of data via a text file. The addition of a function to automatically create a graph of data from the optimization process could also benefit and show instant results instead of taking data from a file and manipulating it yourself. The benefit of storing data into a file to then be manipulated was for the purpose of this project to look back at all the data once it had been stored, but if you were looking at the program from a perspective of looking at performance from the algorithm running, a graph showing up the program had been complete would be a good idea.

To further show the movement of the particle swarm optimization a 3-D model of a graph showing the positions and velocities of each particle with their solution could be implemented therefore over the duration of optimization. I believe this would be a great addition to see the movement itself as the particle’s fitness grow better over time. This would increase runtime due to the movement of the 3-D model taking up computational time therefore results may vary, due to optimization results being better with a faster computational time overall.

## Reflection

The research gave some great understanding to particle swarm optimization, Q-Learning, adaptive selection and the 0-1 MKP itself and was beneficial to the design and the implementation sections, finding past implementations of similar implementations using particle swarm optimization on problems such as the 0-1 MKP made me look at the structure of the program and made me make decisions on how the overall product would connect and perform.

The requirements made me understand the direction in which I needed to follow as I went through each sprint and each design process. I believe that all these requirements were met at the highest level possible, and I am pleased at how this project unfolded and the final product implemented. Areas such as the tabular Q-Learning implementation at first looked difficult but after the literature review looking at areas of Q-Learning and how it can be implemented, I discussed and brainstormed ideas on how the information could be classified within the table which eventually became the idea for the pseudocode designed.

## Feedback

In the first meetings with Mehmet, we discussed the problem of the 0-1 MKP so that was fully understandable, we then progressed into the application of particle swarm optimization into this problem, then I proceeded to do my research and to design ideas and notes as I was doing my research to use in creating pseudocode etc. As meetings progressed, pseudocode and ideas would be brainstormed every meeting or submitted before a meeting to get some feedback on how it could be changed or if it were good to start implementing to get some idea of how it would produce results. This process repeated throughout simple PSO implementation, mutation, adaptive selection and Q-Learning implementation. I was given great feedback with my requirements and design diagrams, stating what was expected of each one and the main areas to show per diagram.

During implementation, many different versions of each implementation (simple PSO, multiple mutation, adaptive selection, Q-Learning) were all shown to Mehmet with feedback to certain areas which did not look correct but had the correct idea and only a few changes were necessary. During the feedback, no major changes were necessary which I was content with. Feedback after the simple mutation implementation therefore led to the idea of having a multiple mutation operator firstly selected by a random number generator and that then lead to the idea of using adaptive mutation selection.

The feedback provided by my supervisor on the project in progress day was positive and highlighted areas where the figures I have created were impressive and where to improve and what to continue with. Overall, the feedback was good and little changes were needed and ideas were brainstormed on how to improve further.

# Conclusion

The aim for this project was to implement a particle swarm optimization algorithm with reinforcement learning to solve a chosen optimization problem in the 0-1 multidimensional knapsack problem. Versions of the algorithm created throughout implementation were saved to compare performance. The algorithm allows the user to select from a range of data sets to which the algorithm then optimizes the data inputted, these results can be compared with the other versions of the algorithm to display as implementation advanced, so did the overall performance of the algorithm in relation to the generations taken to find optimum solutions within data sets and the percentage of optimum solutions reached over an average number of runs. From the testing results shown and the data produced, a conclusion can be made that the implementation of the particle swarm optimization with reinforcement learning was a success.

The project itself has furthered my knowledge into the areas of biocomputing, especially in the areas of swarm algorithms and reinforcement learning. The project has also allowed me to look at areas such as probability matching within optimization. The algorithm development process was a major part of this project and allowed me to enhance my skills in these areas where I could use them in future projects.

## Future Work

To further progress this project, the areas previously mentioned could be added, such as the addition of neural networking within Q-Learning. Functions to input users own data sets could increase the range of data sets the problem could optimize by a simple function to take an input file through a directory. Optimum values could also be removed from some data sets and a function can be added to calculate the optimum value from the knapsack, objects, sizes and weights given through a simple mathematical algorithm.

# 10. References

# 10.1 Literature

Arin, A. Rabadi, G. (2017) *Integrating Estimation of Distribution Algorithms versus Q-Learning Into Meta-Raps for Solving the 0-1 Multidimensional Knapsack Problem. Computers & Industrial Engineering.* 112, pp. 706-720. Available from: <https://www.sciencedirect.com/science/article/pii/S0360835216304077> [Accessed 07 February 2021]

Arora, R.K. (2015­), Optimization: Algorithms and Applications[online], CRC Press. [Accessed 19 November 2020]

Arsham, H. University of Baltimore (Unknown, Last Updated: 2015) *Deterministic Modeling: Linear Optimization with Applications.* Available from: <http://home.ubalt.edu/ntsbarsh/opre640a/partviii.htm#rop> [Accessed 06 February 2021]

Azad, M.A.K., Rocha, A.M.A.C. and Fernandes, E.M.G.P, (2014) Improved Binary Artificial Fish Swarm Algorithm For the 0–1 Multidimensional Knapsack Problems. Swarm and Evolutionary Computation [online]. 14 [Accessed 03 December 2020].

Beasley, J. (1990, Last Updated: 02/2018) "OR-Library: distributing test problems by electronic mail", Journal of the Operational Research Society 41(11) (1990) pp. 1069-1072. [Accessed 10 December 2020].

Boyer, V., Elkihel, M. and El Baz, D. (2009) Heuristics For the 0-1 Multidimensional Knapsack Problem. European Journey of Operational Research [online]. 199 (3) [Accessed 19 November 2020].

Ciaburro, G. (2018) Keras Reinforcement Learning Projects [online], Packt Publishing. [Accessed 19 November 2020].

Giftson Samuel, G. and Christober Asir Rajan, C. (2015) Hybrid: Particle Swarm Optimization-genetic Algorithm and Particle Swarm Optimization-shuffled Frog Leaping Algorithm For Long-term Generator Maintenance Scheduling. International Journal of Electrical Power & Energy Systems [online]. 65, pp. 432--442. [Accessed 22 November 2020].

Haddar, B., Khemakhem, M., Rhimi, H. and Chabchoub, H. (2016) A Quantum Particle Swarm Optimization For the 0-1 Generalized Knapsack Sharing Problem. Natural Computing [online]. 15 (1) [Accessed 22 November 2020].

Hassanien, A.E and Emary, E. (2016), Swarm Intelligence: Principles, Advances and Applications. [online], CRC Press. [Accessed 19 November 2020]

Paul, S. FloydHub (2019) *An Introduction to Q-Learning: Reinforcement Learning.* Available from: <https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/> [Accessed 04 February 2021]

Punchinger, J., Raidl, G.R. and Pferschy, U. (2010) The Multidimensional Knapsack Problem: Structure and Algorithms. Informs Journal on Computing [online]. 22 (2), pp. 250-265. [Accessed 22 November 2020].

Ravichandiran, S. (2018), Reinforcement Learning with Python [online], Packt Publishing. [Accessed 19 November 2020]

Sewak, M. (2019), Deep Reinforcement Learning: Frontiers of Artificial Intelligence [online], Springer. [Accessed 19 November 2020].

Sharma, M. Lopez-Ibáñez, M. and Kazakov, D. (2018) *Performance Assessment of Recursive Probability Matching for Adaptive Operator Selection in Differential Evolution. In: 15th Intl Conf. on Parallel Problem Solving from Nature.* Springer-Verlag, Berlin, 2018. LNCS pp. 321-333. Available from: <https://eprints.whiterose.ac.uk/135483/1/paper_66_1_.pdf> [Accessed 07 February 2021]

Tmats (2017) knapsack\_pso. Available from: <https://github.com/TMats/knapsack_pso> [Accessed 07 December 2020].

Violante, A. Towards Data Science (2019) *Simple Reinforcement Learning: Q-Learning.* Available from: <https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56> [Accessed 04 February 2021]

# 10.2 Software and Libraries

Astah UML (2021) [Computer Program]. Available from: <https://astah.net/products/astah-uml/> [Accessed 10 October 2020].

Lucidchart (2021) [Computer Program (Online)]. Available from: <https://www.lucidchart.com/pages/> [Accessed 15 October 2020].

Matplotlib (2021) [Computer Program]. Available from: <https://matplotlib.org/> [Accessed 10 January 2021].

numPy (2020) [Computer Program]. Available from: <https://numpy.org/> [Accessed 04 November 2020].

Python 3.9 (2020) [Computer Program]. Available from: <https://www.python.org/> [Accessed 20 September 2020].

TeamGantt (2020) [Computer Program (Online)]. Available from: <https://www.teamgantt.com/> [Accessed 05 October 2020].

# 11. Appendices

## 11.1 Appendices 1: Data Sets

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Set** | **Knapsacks** | **Objects** | **Optimum Value** |
| 1 | 2 | 105 | 1095445 |
| 2 | 4 | 28 | 3418 |
| 3 | 4 | 35 | 3186 |
| 4 | 5 | 30 | 4554 |
| 5 | 5 | 40 | 5557 |
| 6 | 5 | 50 | 5643 |
| 7 | 5 | 50 | 6159 |
| 8 | 5 | 60 | 8633 |
| 9 | 5 | 70 | 9580 |
| 10 | 5 | 80 | 8947 |

Table 4: Data Sets

## 11.2 Appendices 2: Extraction of Q-Table from Implementation

|  |  |  |
| --- | --- | --- |
| **Reward** | **Old Particle State** | **New Particle State** |
| -1.0 | [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0] [0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0] [0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1] [1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1] [0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1] 11477 | [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0] [0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0] [0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1] [1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1] [0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1]  11718 |
| 1.0 | [0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1] [0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0] [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1] [0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1] [0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0] 12338 | [0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1] [0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0] [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1] [0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1] [0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0] 11374 |
| 1.0 | [1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] [0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0] [0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1] [0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0] [1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0] 11758 | [1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] [0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0] [0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1] [0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0] [1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0] 11331 |
| 1.0 | [1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0] [1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0] [1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0] [0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0] [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0] 9940 | [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0] [1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0] [1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0] [0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0] [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0] 9781 |
| 1.0 | [0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1] [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0] [1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1] [1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0] [0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0] 11707 | [0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1] [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0] [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1] [1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0] [0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0] 11208 |

Table 5: Extraction of Q-Table

## Appendices 3: Testing Results



*Table 6: Percentages of Optimum and 1% of Solution Reached*