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. My 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

# Table of Contents

[Abstract 2](#_Toc67858977)

[Acknowledgements 3](#_Toc67858978)

[Table of Contents 4](#_Toc67858979)

[Table of Figures 5](#_Toc67858980)

[Table of Tables 6](#_Toc67858981)

[1. Introduction 7](#_Toc67858982)

[2. Literature Review 7](#_Toc67858983)

[2.1 Previous Projects 12](#_Toc67858984)

[2.2 Identified Gaps 14](#_Toc67858985)

[2.3 Project Scope 14](#_Toc67858986)

[3. Requirements 15](#_Toc67858987)

[4. Methodology 16](#_Toc67858988)

[4.1 Planning 16](#_Toc67858989)

[4.1.1 Sprint 1 16](#_Toc67858990)

[4.1.2 Sprint 2 17](#_Toc67858991)

[4.1.3 Sprint 3 17](#_Toc67858992)

[5. Design 17](#_Toc67858993)

[5.1 Class Diagram 18](#_Toc67858994)

[5.2 Activity Diagram 19](#_Toc67858995)

[5.3 Pseudocode 20](#_Toc67858996)

[5.4 Interface 24](#_Toc67858997)

[6. Implementation 24](#_Toc67858998)

[6.1 Sprint 1 (Particle Swarm Optimization) 25](#_Toc67858999)

[6.2 Sprint 2 (Q-Learning algorithm) 29](#_Toc67859000)

[6.3 Sprint 3 (Testing with Data Sets and Data Collection) 31](#_Toc67859001)

[7. Results and Findings 32](#_Toc67859002)

[References 32](#_Toc67859003)

[Appendix 33](#_Toc67859004)

# Table of Figures

[Figure 1: Reinforcement Learning Flow. 9](#_Toc67408900)

[Figure 2: Q-Learning Bellman Equation 10](#_Toc67408901)

[Figure 3: Multidimensional Knapsack Problem (MKP) 11](#_Toc67408902)

[Figure 4: Gantt Chart 17](#_Toc67408903)

[Figure 5: Class Diagram 19](#_Toc67408904)

[Figure 6: State Diagram 19](#_Toc67408905)

[Figure 7: Initialization Pseudocode 20](#_Toc67408906)

[Figure 8: PSO Pseudocode 21](#_Toc67408907)

[Figure 9: Adaptive Selection Operator using Probability Matching Pseudocode 22](#_Toc67408908)

[Figure 10: Q-Learning Pseudocode 23](#_Toc67408909)

[Figure 11: Example of data used during Implementation (w/ Description) 24](#_Toc67408910)

[Figure 12: PSO altering algorithm. 25](#_Toc67408911)

[Figure 13: Single Mutation algorithm. 26](#_Toc67408912)

[Figure 14: Multiple Mutation algorithm. 26](#_Toc67408913)

[Figure 15: Adaptive Selection algorithm. 27](#_Toc67408914)

[Figure 16: Q-Learning algorithm. 29](#_Toc67408915)

# Table of Tables

# 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 my 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 my project is to understand how the agent in my case particles, react with the environment to alter their data, which then leads to a change of fitness. Reinforcement learning can teach my 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 was 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. This allows me to study further into areas such as Q-Learning which I can use in my project 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.

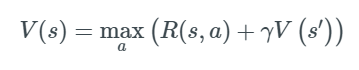


Figure 2: Q-Learning Bellman Equation

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 allow me to understand the mathematics and the understanding of 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, therefore in my project, using Q-Learning should allow me to get significantly better fitness’s through its algorithm.

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. This is something I want to explore creating in my own project to solve my multi-dimensional knapsack problem, I will look for research in previous work of using PSO and RL in binary problems. This paper allows me to understand the underlying concept of particle swarm optimization and how I can use it in my project, it allows me to understand how I can put 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:

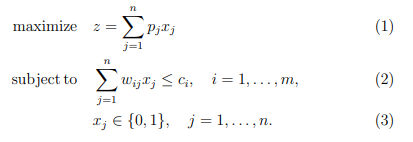


Figure 3: Multidimensional Knapsack Problem (MKP)

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 my research so far allowed me to research into topics that further optimized my solutions, such as operators to add on to my 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). In my project, I plan 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 my project, it allows me to understand how I can choose mutations based off its past behavior and through probability matching this should show a increase in performance in getting to an optimal solution.

This operator will be one of the main operators of my 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.

## 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, I initially decided to use this framework but then changed to design and program from scratch from my own plan, but the idea and implementation is very similar. 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 allows me to understand the connection between reinforcement learning and the other techniques but allows me to gather information on how I can connect Q-Learning to my PSO.

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 I want to do, 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.

I found an implementation of Particle Swarm Optimization on the 0-1 MKP written in Java, by author TMats on GitHub created in 2017, reading through the code and understanding what each function, class was doing allowed me to have a better understanding of how PSO works with this problem, it will also allow me to compare against pseudocode I have written to make sure how I am going to implement my algorithms is correct. Having a clear understanding of the problem ahead after reading this piece of code was helpful in going forward with my planning and implementation.

## 2.2 Identified Gaps

Many projects that I 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, allowing me to look at areas where I could find an area 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 my 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), I decided to go for an area that already has existing findings in a simple particle swarm optimization and the 0-1 MKP but further research into this by adding multiple different operators and a reinforcement learning algorithm.

The gap between my project and other projects I have 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, I have not been able to find particle swarm optimization with reinforcement learning using these 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

There are 6 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. |
| 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

I have decided to use the Agile method while implementing my project, as this allows for me to receive feedback especially if I come across problems during the implementation period, which I am certain will arise. This feedback will allow me to make changes in the correct areas and raise new ideas on how I can implement my project.

I have decided to use Sprints to spread out how I implement my project, which I will show below in my planning, I decided to use sprints because of the experience I have in using them, it also allows me to work to certain deadlines where I can get feedback after these deadlines and can go back to finalize areas I have worked on. Overall, I have a good understanding of using sprints and I believe this to be the optimal way for me to do my project.

After each sprint, I will aim to get feedback on what I have produced and attempt to make slight changes in accord to the feedback that I have been given, and finally “clean up” any code that I have created.

## 4.1 Planning

I have used TeamGantt to create my 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. I would change this during the project to accommodate any changes I have done in the order of my implementation / testing or any other action. Figure 4 is my complete Gantt Chart once development has been completed.

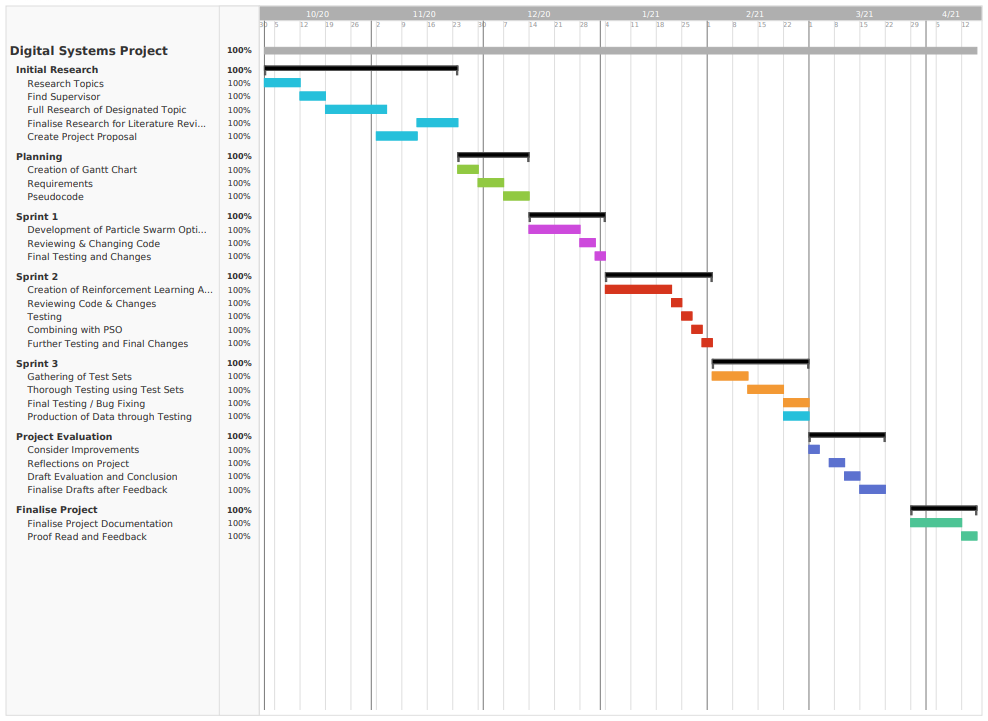


Figure 4: Gantt Chart

### Sprint 1

The first Sprint is designed to develop a particle swarm optimization (PSO) algorithm, which is the base of the program, this will allow me to then set up a reinforcement learning algorithm to implement the reward system dependent on the particle swarm optimization’s state. During this period, I will be creating a multiple mutation operator with an adaptive selection operator. This 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 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 and allow me to return to make changes wherever necessary.

### Sprint 3

The final sprint is to make sure my finished program containing the PSO and RL algorithms fully optimize the 0-1 MKP. This will be done through extensive testing using multiple test data acquired from Brunel University and might require changes therefore a whole sprint will be required to make sure the full algorithm works correctly; the final part of the sprint will be designated to bug fixing if there are any and final changes and testing and creating test cases.

# Design

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

The design of my 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 my 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 my algorithm and how different objects data is manipulated in different ways.

## 5.1 Class Diagram

I used Astah UML (2020) to design my class diagram before I went on to design my pseudocode, I had a plan 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, I created classes 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.

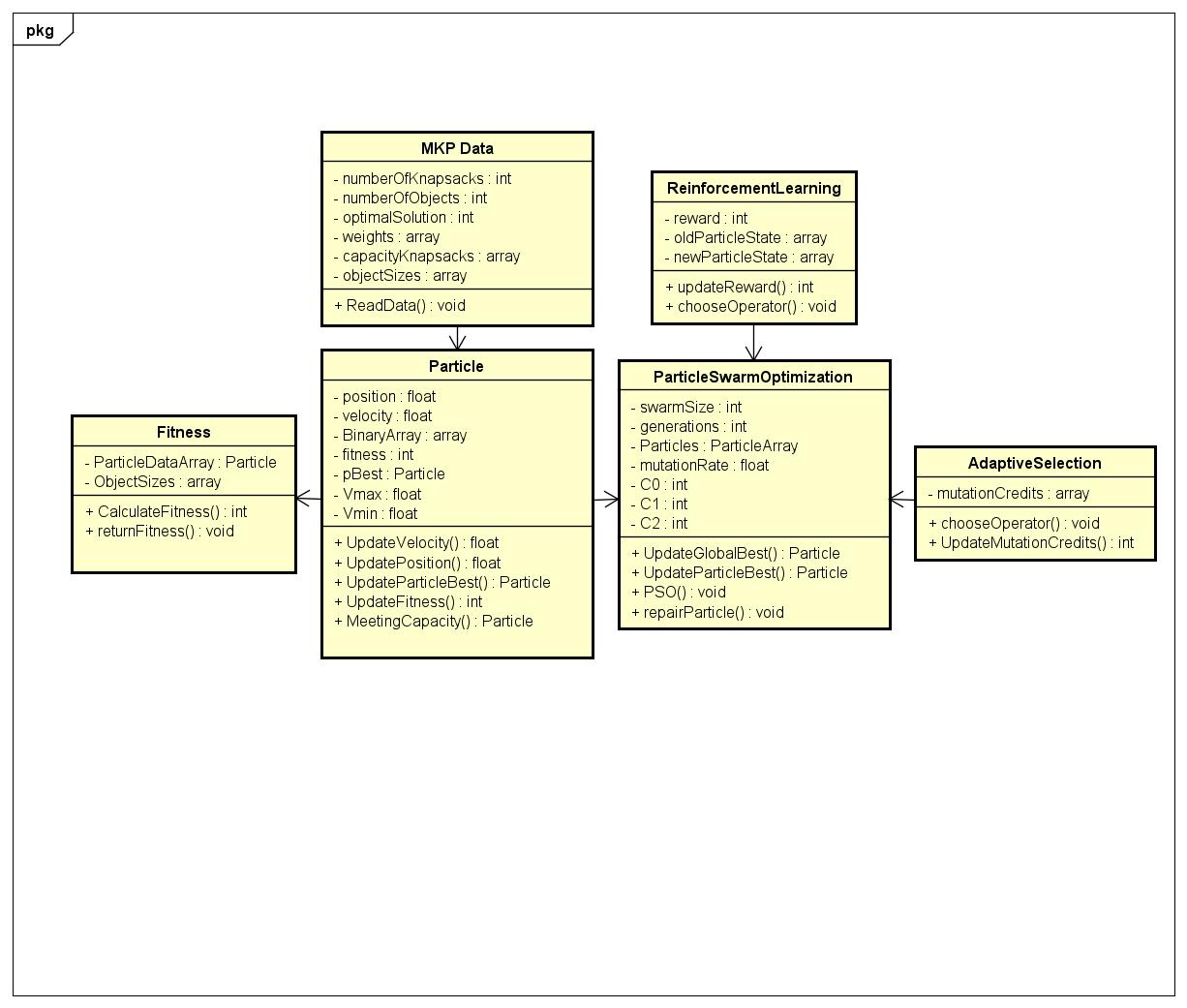


Figure 5: Class Diagram

## 5.2 Activity Diagram

For my activity diagram, I used Lucid.app (2021) to design the activities I wanted my 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 me to look at areas 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, this allowed me to create pseudocode for the PSO created in steps as described in the class diagram of having initialization then optimization functions. To plan pseudocode for the initialization I had to decide how I would create these initial solutions. 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 a 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, I designed each particle to be shown as [0,1,…][1,0,…], fitness, position, velocity. Therefore, when all particles are stored in a whole array, I can simply iterate through each particle and the indexes would all be the same e.g. 0 for the binary data, 1 for fitness etc. I would create valid initial solutions, 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 there 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, I designed my final pseudocode for initialization which was ready for implementation.

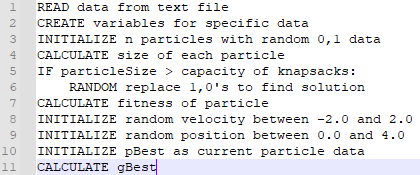


Figure 7: Initialization Pseudocode

I could then move onto designing pseudocode for my PSO, 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. It allowed me to design many different drafts of how I wanted to implement the PSO, with discussion also in my supervisor I was very quickly able to create pseudocode for how I wanted my PSO to look. My PSO would iterate through generations, and I would create 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. I wanted my program to iterate through each particle in each generation, changing data randomly that was like the global best, I would then mutate this particle allowing it to optimize further, if it fit the constraints and the capacities, I would then move on to class it as a valid solution, updating velocity, position and then repeating for the next particle in the swarm. At the end of each generation, I wanted the best particle fitness to 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.

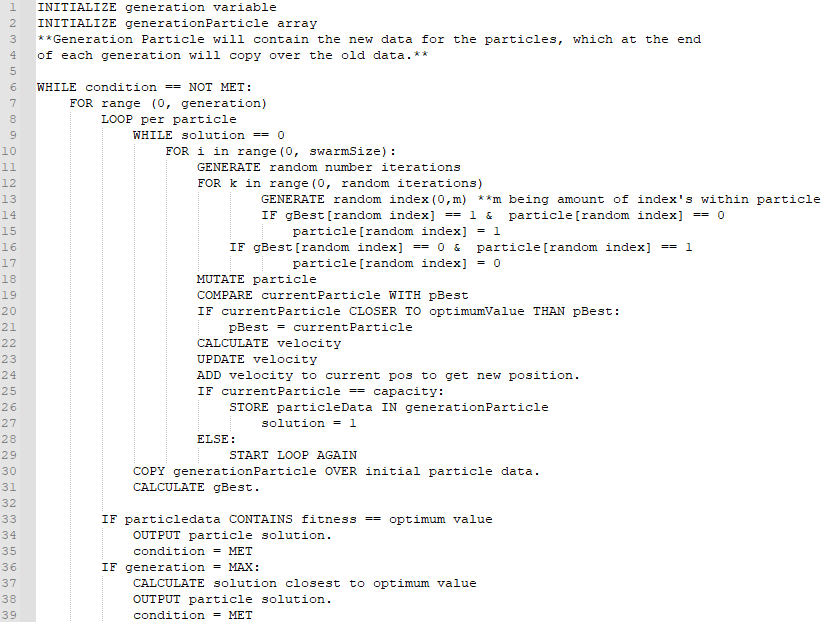


Figure 8: PSO Pseudocode

Once I had the basic design for my PSO with a single mutation operator, 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.

I needed to then design my adaptive selection operator 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, I designed an idea 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.

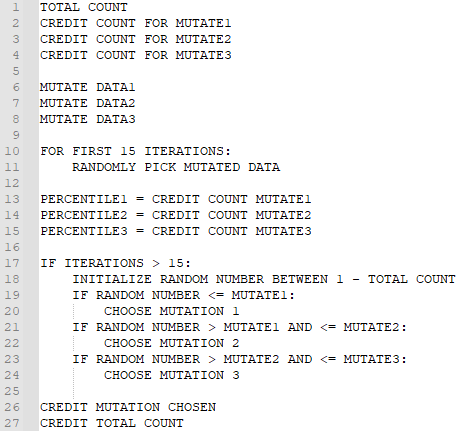


Figure 9: Adaptive Selection Operator using Probability Matching Pseudocode

To make comparisons to the adaptive selection with probability matching, I have designed a Q-Learning algorithm 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.

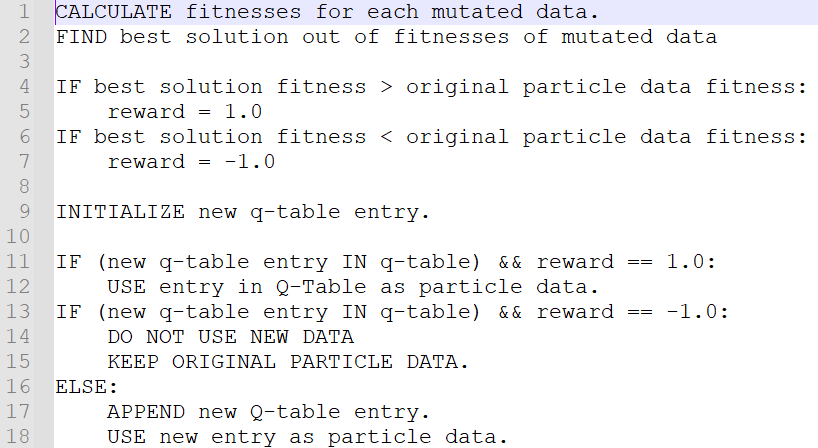


Figure 10: Q-Learning Pseudocode

## 5.4 Interface

For the interface for my program, 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, position, velocity 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 I used the sprints I had created during the planning stage, this therefore split the implementation into three sub-implementations: the PSO, the reinforcement learning and finally, the full optimization of the 0-1 MKP and testing.

## 6.1 Sprint 1 (Particle Swarm Optimization)

My 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 allowed me 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, I created data which I was able to take from the test data sets from the Brunel OR-Library, created by J E Beasley (1990), I then proceeded to take this data and sort it, so it was easier to read directly into variables, arrays etc.

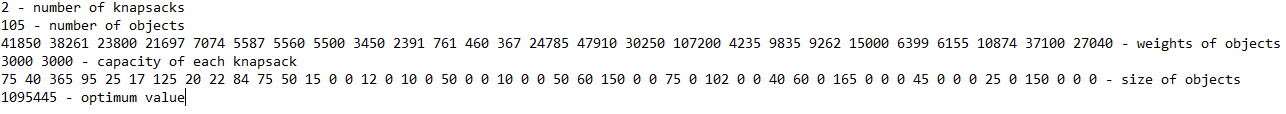


Figure 11: Example of data used during Implementation (w/ Description)

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 I initialized random 0,1’s in an array for each particle. I then created functions 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, I needed the sizes of the knapsacks created to equal the capacity of each knapsack, to do this I created a function 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 allowed me to edit the population within a different variable and overwrite it at the end of every generation, this was in case any errors went on during the PSO which affected data. I split creating the PSO into 4 different pieces, one using a simple PSO algorithm with no operators, the second using the first algorithm but with a single mutation, the third continuing from the second piece having 3 different mutation operators and a random selection scheme and finally the last piece, having a PSO with 3 different mutation operators and an adaptive selection operator.

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 I am dealing with 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.

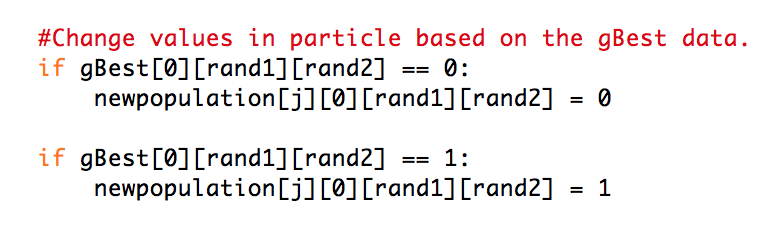


Figure 12: Particle altering algorithm.

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 that data is written into a CSV file, to store the best data for each generation and to calculate the best solution found once max generations has been reached.

For the single mutation operator, I simply added a bit flip mutation operator which just 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. I then added two more mutation operators, 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.

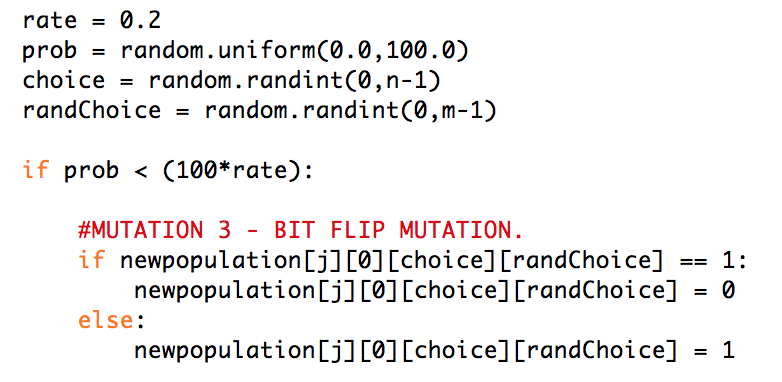


Figure 13: PSO Single Mutation algorithm.

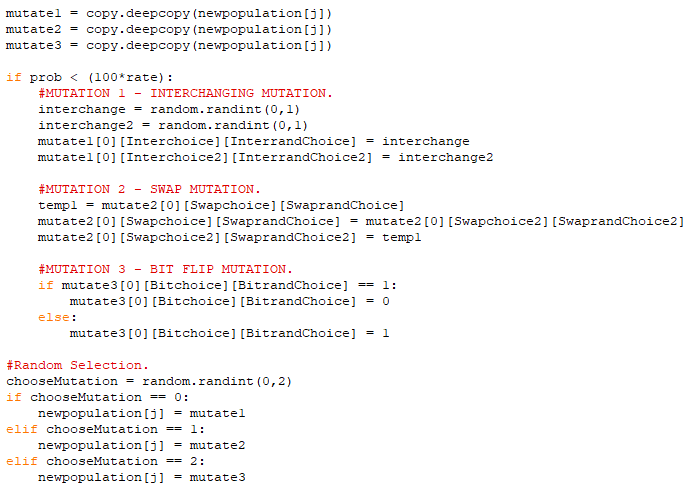


Figure 14: Multiple Mutation algorithm.

I improved the random number generator 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.

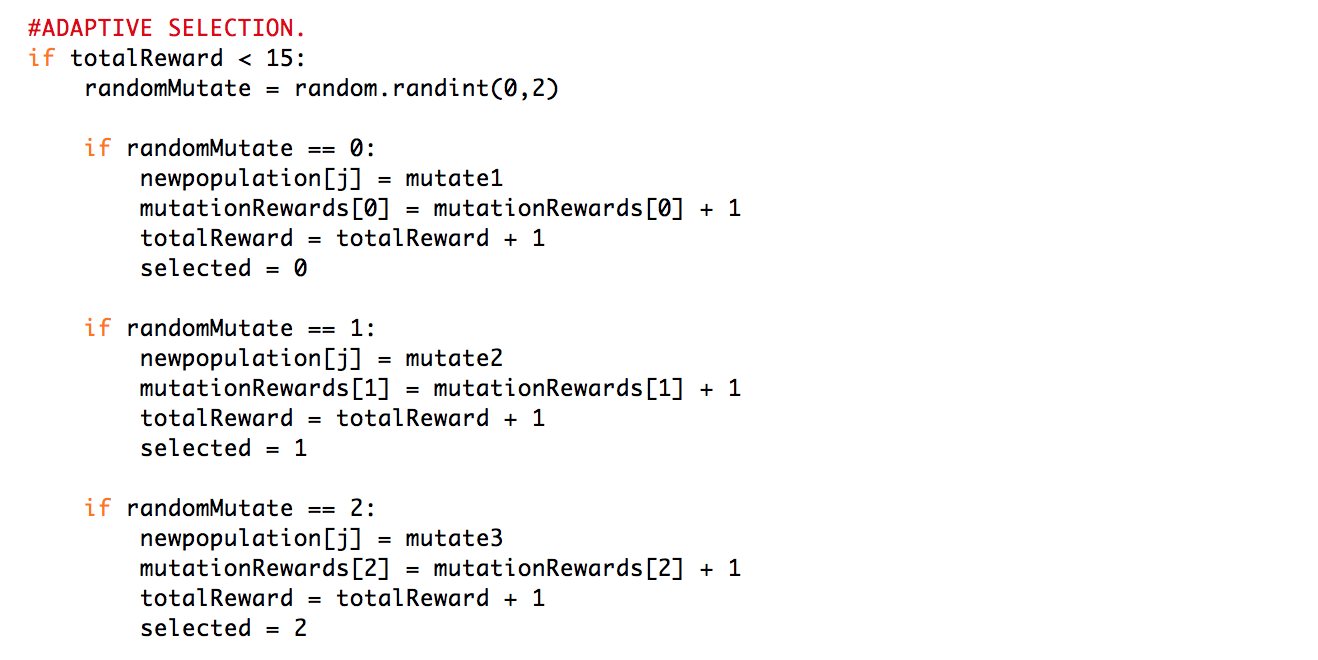




Figure 15: Adaptive Selection algorithm.

At the end of this sprint, the aims and objectives of having a PSO with multiple random mutation and adaptive selection using probability matching, as well as a simple PSO by itself have all been met. I have tested the algorithms 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.

## 6.2 Sprint 2 (Q-Learning algorithm)

For the implementation of the 2nd 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, I initialized the Q-Learning table 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 my 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 my 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 appendix.

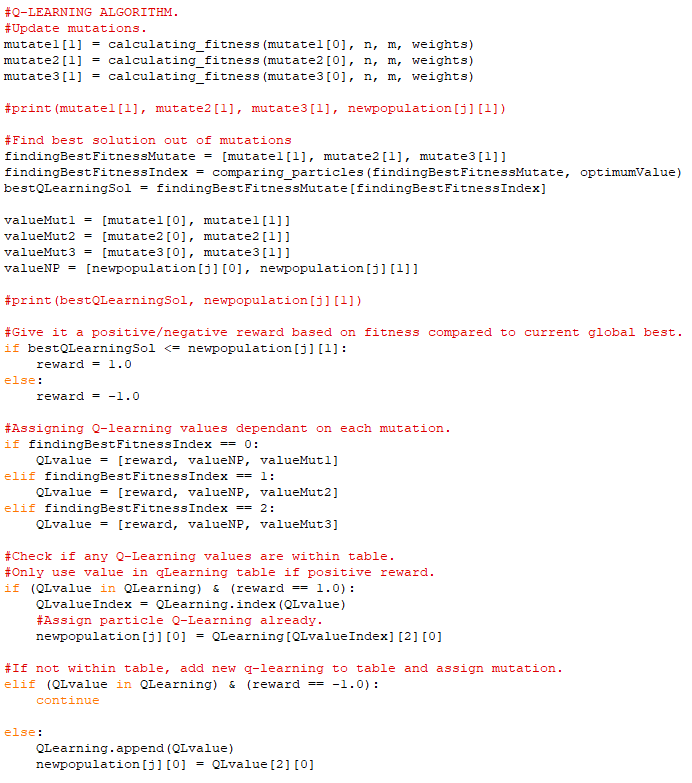


Figure 16: Q-Learning algorithm.

## 6.3 Sprint 3 (Testing with Data Sets and Data Collection)

For Sprint 3, my aim was to test my implemented algorithms, with 10 different test sets to collect data to use for my 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 my 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.

I also created graphs to prove that my adaption mutation selection operator successfully worked during implementation. 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 a working adaptive selection operator.

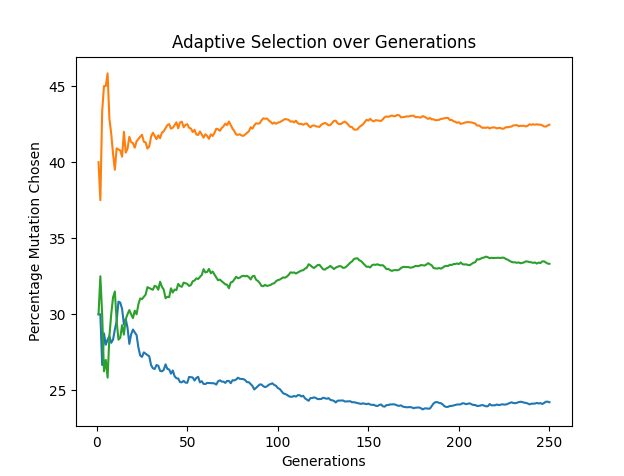
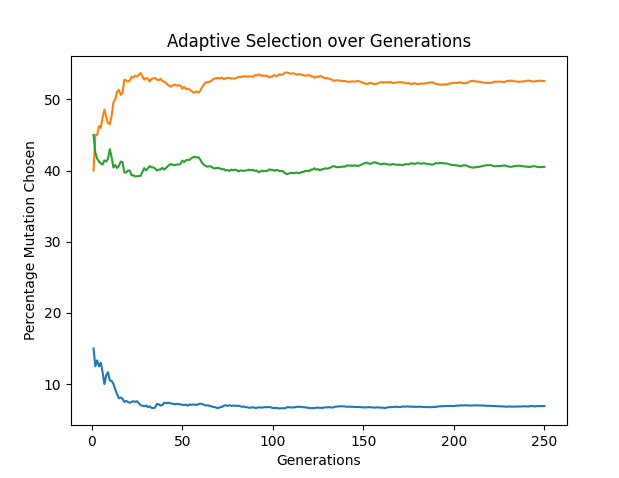


Figure 17 and 18: Adaptive Selection Graphs

The aims and objectives that will be completed from the data collection will be the comparison of performance from the algorithms created and will show the progression in optimization throughout my project from the starting standard PSO algorithm with single mutation, to the final developed PSO with Q-Learning. The creation of the graphs for the adaptive selection also confirms a working adaptive selection algorithm.

# 7. Results and Findings

**SHOW GRAPHS AND TABLES OF DATA FROM TESTING.**

# 

# 9. References

# 9.1 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]

# 9.2 Software TO DO.

Astah UML

Lucid.app

Matplotlib

numPy

Python 3.9

TeamGantt

# 10. Appendix

# 10.1 Appendix 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 2: Data Sets

# Appendix 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 3: Extraction of Q-Table