Digital Systems Project

**To be done:**

**Class diagram/sequence diagram with RL**

**Psuedocode to be added for RL.**

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 binary operators which will run through a reinforcement learning driven selection scheme 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. (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 driven selection to automatically select the swarm intelligence algorithms operators, such as mutation, to solve binary problems.

# Acknowledgements

# Table of Contents

[Abstract 2](#_Toc64035760)

[Acknowledgements 3](#_Toc64035761)

[Table of Contents 4](#_Toc64035762)

[1. Introduction 5](#_Toc64035763)

[2. Literature Review 6](#_Toc64035764)

[2.1 Previous Projects 8](#_Toc64035765)

[3. Requirements 9](#_Toc64035766)

[4. Methodology 10](#_Toc64035767)

[4.1 Planning 11](#_Toc64035768)

[4.1.1 Sprint 1 11](#_Toc64035769)

[4.1.2 Sprint 2 11](#_Toc64035770)

[4.1.3 Sprint 3 12](#_Toc64035771)

[5. Design 12](#_Toc64035772)

[5.1 Class Diagram 12](#_Toc64035773)

[5.2 Sequence Diagram 13](#_Toc64035774)

[5.3 Pseudocode 13](#_Toc64035775)

[5.4 Interface 15](#_Toc64035776)

[6. Implementation 16](#_Toc64035777)

[6.1 Sprint 1 16](#_Toc64035778)

[6.2 Sprint 2 16](#_Toc64035779)

[6.3 Sprint 3 16](#_Toc64035780)

[References 16](#_Toc64035781)

# 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 the selection scheme, 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 program.

# 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).

Diagram

Description automatically generated

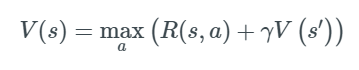
*Figure 1.0: Reinforcement Learning Flow.*

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 1.1 Q-Learning Bellman Equation.*

Paul (2019) explains the equation where:

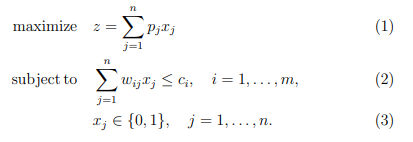
* s = a particular state
* a = action
* s’ = state to which the agent goes to from s
* 𝜸 = discount factor
* R(s,a) = a reward function, taking inputs of a state (s) and action(a) and outputs a reward value e.g. 0 or 1.
* V(s) = value of being in a state

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.

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.

Punchinger et al. (2010) researched the structure and algorithms of the MKP and as shown in Figure 1.1 that the MKP can be defined as:



*Figure 1.2: 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 1.1. 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.

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 1.1 (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.

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”. There 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.

### 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.

## 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 selection 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 6 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.

|  |  |
| --- | --- |
| 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 should have a clear structure on how it is solving the 0-1 MKP. |
| NF2 | The program should retrieve the data quickly. |
| NF3 | The program should be able to run on any OS. |
| NF4 | The program should present how it is solving the data by every time there is a generation. |
| NF5 | The program should show how long it took to solve the problem. |
| NF6 | The program could write its information into csv files/text files to show in different runs how they compare in performance, this also allows for the creation of graphs. |

*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 1 is my complete Gantt Chart once development has been completed.

(Complete gantt chart.) – At finish.

*Figure 2: 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 operators for the optimization, e.g. swarming, selection, mutation. This will then be tested on test cases 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, extensive testing will also be needed so I am able to move onto the next sprint.

### 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 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.

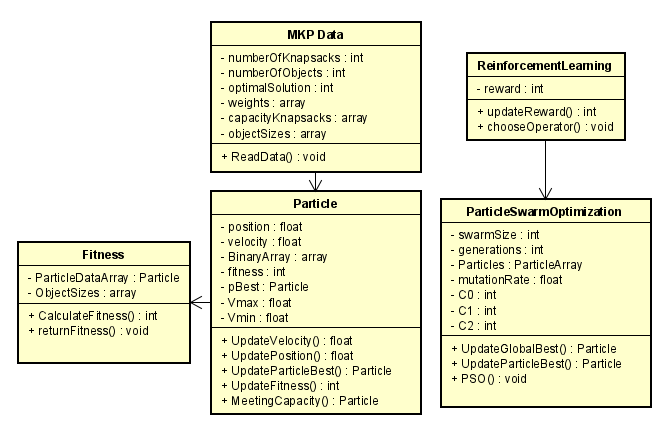
## Design

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.

### 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 operators to generate new solutions. Figure 3 represents the class diagram.

*This could be changed. (RL would need updating.)*



*Figure 3: Class Diagram*

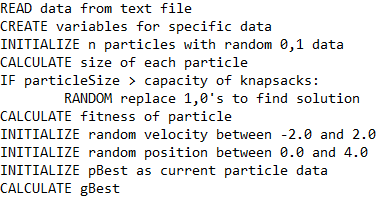
### 5.2 Sequence Diagram

---NOT COMPLETE.

*RL sequence diagram – if necessary.*

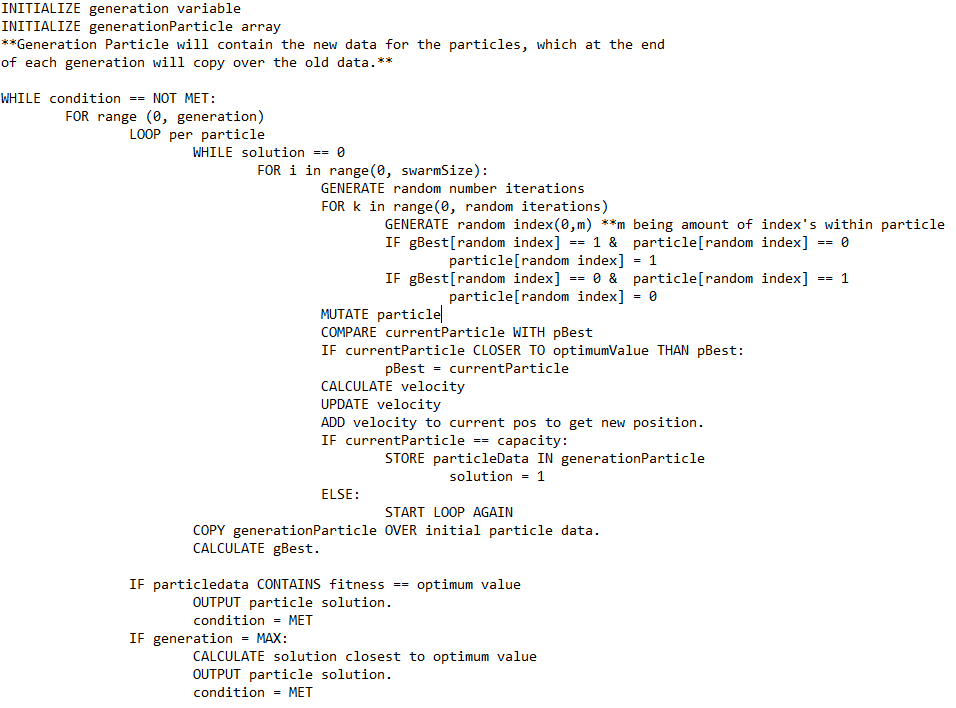
### Pseudocode

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 4.1 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 similar to 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 4.2 PSO Pseudocode*

*RL pseudocode. (To be done.)*

### Interface

For the interface of my program, as my program will be able to run many different problems of the 0-1 MKP if the data in the text file is laid out correctly, it will display the optimal fitness at the start of the run. Once the PSO begins**,** it would output generations of the PSO returning the best fitness in the swarm, this allows to see the initial best solution and the progress of the optimization over time. Once the run has been complete, the time taken to reach the optimal solution and the number of generations will be outputted, if max generations has been reached, it will state that the max generations has been reached.

*Probably going to be changed over time.*

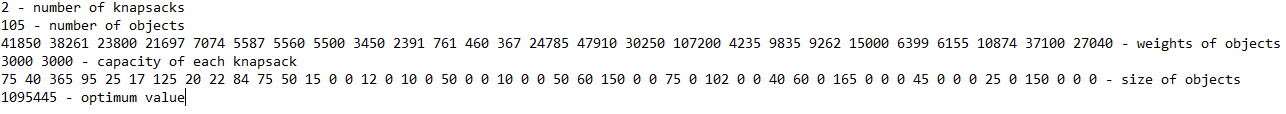
## Implementation

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.

### Sprint 1 (Particle Swarm Optimization)

All of my project will be built using Python, some external directories were needed such as numPy, which 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 5.1 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. However, it would only initialize it to a 0 or 1 if the size of that slot was greater than 1. 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. *NOT COMPLETE.*

### Sprint 2

-------

### 6.3 Sprint 3

# References