Genetic Algorithms for Multi-knapsack balance problem

Main practical assignment

Guillermo Diz Gil

Carmen Mª Muñoz Pérez

Tutor: Agustín Riscos Núñez

Index

Genetic algorithm

To carry out this assignment, before analysing and modelling the multi-knapsack balance problem, a library has been developed to facilitate the resolution of genetic algorithm problems. The code has very detailed explanations to make it easier to understand. Some relevant aspects of certain implementations that have been carried out will be detailed below:

Combining individuals

There are different ways of obtaining chromosomes by combining other individuals. In all of them there will be two randomly chosen parents from which a new chromosome will arise. There could be a probability of crossover, but we have decided that it will always be done, i.e., it will always have a 100% probability.

It should be noted that normally when such combinations of two individuals are made, two offspring chromosomes are usually the result. In our case, it has been decided to obtain one offspring for each of the two parents. This is due to reasons such as less complexity in the implementation and the fact that it is easier to obtain a generation with an odd number of chromosomes.

The different types of crossovers that have been implemented will be detailed below:

Single point crossover

A random crossover point will be chosen and applied to both parents. Thus, each parent will be divided into two parts. The new chromosome will be formed in the following way: all the genes before the crossover point are inherited from first parent, while all the genes after the crossover point are inherited from the second parent.

It should be noted that the chromosome length must be at least two genes long.

Triple point crossover

Three random crossover points will be chosen and applied to the two parents. Thus, each parent will be split into four parts. The new chromosome will be formed in the following way: the even segments of genes will be inherited from the first parent, while the odd segments of genes will be inherited from the second parent.

It should be noted that the chromosome length must be at least six genes long.

Uniform crossover

This time there is no crossover point. Thus, for each position in the offspring, it will be randomly chosen which parent inherits from, i.e, each gene is chosen randomly either from the first parent or from the second one.

It should be noted that the chromosome length can be whatever.

Mutations in individuals

Mutation is an operator used to maintain some diversity between the chromosomes of one generation and the next. Thus, there will be a probability of mutation, i.e., the probability that the chromosomes in the population may change. When this mutation occurs, the new chromosome that is generated is randomly selected.

Selection mechanisms

When performing such algorithms, there will be occasions when certain individuals must be chosen from a population.

For this purpose, there are several selection mechanisms that normally favour those chromosomes with better fitness, although there is always a random component.

The different types of selection mechanisms that have been implemented will be detailed below:

Roulette wheel selection

This selection method is part of fitness-proportional selection, which consists of random selection but giving a higher probability to chromosomes with higher fitness. So, each chromosome has a probability of being selected proportional to its fitness value, which is calculated using the given fitness function. Moreover, this selection method can only be used when we have a maximization problem.

In the roulette wheel selection, first we calculate for everyone in the population, its associated cumulative sum of the values of the fitness. After that, a list of random numbers between 1 and the total sum of values is generated. Finally, for each random number the first chromosome whose cumulative sum is greater than or equal to it is returned.

Regarding our implementation, it will not be allowed to use this selection method if it is a minimisation problem. It is still possible that there is some negative fitness or 0 in maximisation so that so it is important to say that non-positive fitness value will be assumed as 1.

It should be noted that a chromosome can be selected multiple times.

Tournament selection

To select an individual with this method, k chromosomes will be chosen from the population and the best one (the one with the best fitness) will be selected. This process will be carried out until the desired number of chromosomes is achieved.

This method can be used in both maximisation and minimisation problems. Moreover, the higher the k, the higher the selection pressure.

It should be noted that a chromosome can be selected multiple times.

Multi-knapsack balance problem analysis

Statement - *Given a list of items, where each item has a weight associated with it, and given a number n, the problem is to find a balanced distribution of the objects into n subsets, in such a way that they all have the same weight (or as similar as possible)*.

Once the library for solving genetic algorithms has been built, we focus on modelling the multi-knapsack balance problem.

To do so, we first analyse the statement. Once analysed, several interpretations of the problem were considered. These interpretations will be presented in general terms below and we will comment on which one has been chosen.

Alternative 1

There are n subsets, and each subset has a different maximum capacity. In addition, there is a list of items and not all of them must be chosen. Therefore, the aim is to maximise the number of items that are distributed among all subsets, while minimising the difference in the total weight of each subset.

Alternative 2

There are n subsets that have a maximum capacity, and it is the same for all. In addition, there is a list of items and not all of them must be chosen. Therefore, the aim is to maximise the number of items that are distributed among all subsets, while minimising the difference in the total weight of each subset.

Alternative 3

There are n subsets that do not have a maximum capacity. In addition, there is a list of items, and they all must be distributed among the subsets. Therefore, it is intended to minimise the difference in the total weight of each subset.

Initially, when analysing the problem, it was thought that not all items should be taken and that subsets had a maximum weight. But, first, we realised that all the items in the statement must be taken, since it tells you that they are distributed among the subsets and does not specify that any item can be left untaken. On the other hand, it does not specify any restriction about a maximum capacity of the subsets. Furthermore, if there were a maximum capacity, you would have to consider whether all the items fit in the capacity of all the subsets. Therefore, it was determined that the problem would be modelled according to alternative 3.

Codification of the problem

The multi-knapsack balance problem is an optimisation problem since it is intended to minimize the difference between the total weight of each knapsack individually and the perfect average weight that a knapsack should have if all the objects were perfectly distributed along all the knapsacks.

Genes

1,2, 3, … X (with X being the number of knapsacks available)

Individual length

Y (with Y being the number of objects available)

Decode

The positions represent the objects. The genes represent the knapsacks. Each gene at the position i of the chromosome represents the knapsack where the object i has been put into, so every object is put in only and just only one knapsack.

Fitness

We calculate the distance between the total weight of each knapsack individually to the perfect average weight that a knapsack should have if all the objects were perfectly distributed along all the knapsacks, allowing us to minimize the difference between them.

It has not been necessary to add any penalties, as no restrictions are specified. In addition, in the modelling of the problem, all items have been forced to be distributed in the knapsacks, so there is no penalty for not taking an item.

Type

MIN

Analysis of multi-knapsack balance problem instances

Once the problem has been completed, instances have been made with different inputs and by changing some of the parameters to check that they work correctly.

It should be noted that for all the instances there are two parameters that have been the same for all of them: the type of optimisation of the problem and the selection method. This is since this problem has been modelled as a minimization problem, which means that of the two selection methods implemented, only tournament selection can be used (since roulette wheel selection is only for maximization problems).

The study will be divided into two sections:

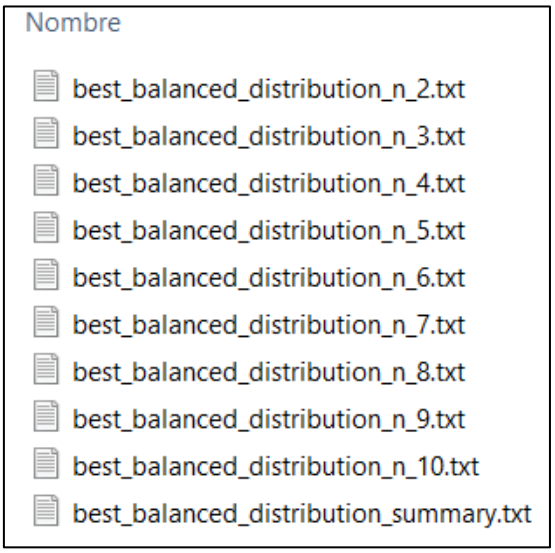
- Firstly, we will analyse instances where all the input data are the same except for the number of knapsacks. This is because the aim is to study, given the same number of objects, with which number of knapsacks the balance distribution is better.
- On the other hand, we want to analyse what happens with instances that have the same objects and the same number of knapsacks as input, but with different parameters such as mutation probability. The main purpose of this is to analyse how these parameters influence the outcome of the problem.

Best balanced distribution according to number of knapsacks

To carry out this study, a specific method has been created. This has been done to be able to change the data more easily and to obtain a more comprehensive analysis of the data. The aim is to study, given the same number of objects, with which number of knapsacks the balance distribution is better.

Firstly, the method will allow to specify the range of n knapsacks to be studied, which in our case will be from n=2 to n=10. It will be randomly specified the number of objects to be used. For this purpose, a random number between 100 and 200 has been searched for, resulting in 155 objects whose weights will be created randomly and will range between 1 and 100. The generated objects will be used for the n knapsacks.

So, we will run the method that will return one file for every n. It will also tell us for which number of knapsacks the balance is better. The data used in this study can be found in the reports folder, specifically in the folder "best_balanced_distribution_19-06-2021_20-32-00".



best_balanced_distribution_19-06-2021_20-32-00 folder

Detailed data for each n can be found in that folder. Each file will indicate for each n which items are in each knapsack with their respective weights, and the total weight for that knapsack. Finally, the fitness, the average expected weight and the weights

finally obtained will be indicated. So, in our case we will have 9 files of this type as we have used n=2, 3, ... 10.

In addition to these 9 files, we will get a summary file. This will contain for each n the distribution obtained in the knapsacks, as well as the fitness. Finally, the best balance obtained and to which n it corresponds will also be indicated. The summary obtained is as shown below:

```
Found solution for n=2: knapsacks [3832, 3834] with distance to perfection 2.0
Found solution for n=3: knapsacks [2553, 2555, 2558] with distance to perfection 12.666666666666666
Found solution for n=4: knapsacks [1917, 1898, 1928, 1923] with distance to perfection 517.0
Found solution for n=5: knapsacks [1508, 1549, 1571, 1523, 1515] with distance to perfection 2748.7999999999997
Found solution for n=6: knapsacks [1252, 1235, 1331, 1289, 1291, 1268] with distance to perfection 5723.333333333333
Found solution for n=7: knapsacks [1109, 1097, 1078, 1027, 1109, 1137, 1109] with distance to perfection 7268.857142857144
Found solution for n=8: knapsacks [963, 953, 923, 949, 990, 975, 935, 978] with distance to perfection 3597.5
Found solution for n=9: knapsacks [879, 864, 822, 894, 828, 806, 878, 818, 877] with distance to perfection 8685.555555555555
Found solution for n=10: knapsacks [792, 761, 765, 780, 757, 764, 702, 765, 798, 782] with distance to perfection 6356.399999999999
============================
BEST solution is n=2
```

Best balanced distribution summary

As can be seen, the best result has been obtained by having two knapsacks. This is because having randomly generated objects is more likely that the distribution between only two knapsacks is always more optimal than having more knapsacks. The more knapsacks there are, the more difficult a balanced distribution will normally be. However, it is normal to get results like the one we have obtained in which having 8 knapsacks is better than having 6 or 7. This is because for this set of objects, due to their weights, for n=8 the distribution will be more optimal.

It should be noted that this study has been carried out several times and the best result has always been obtained for n=2. Therefore, it is concluded that given randomly generated objects, 2 knapsacks will always be the best choice. For larger n will depend, it may be that one n is better than another, but the algorithm has not been able to find it. On the other hand, n=2 is better for random objects, but if a particular population of certain characteristics is chosen, it is possible that another n is better.

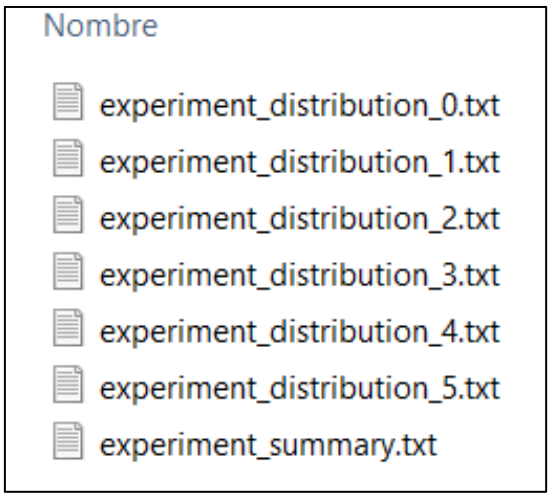
In summary, it has been concluded that given the same number of objects, the best balance distribution will normally be with 2 knapsacks. However, it will depend on the number of objects and their weights, so another number of knapsacks may be the most optimal.

Analysis of instances by changing parameters

A specific method has also been used for this study. This study tries to analyse the influence of certain parameters on the results of the problem.

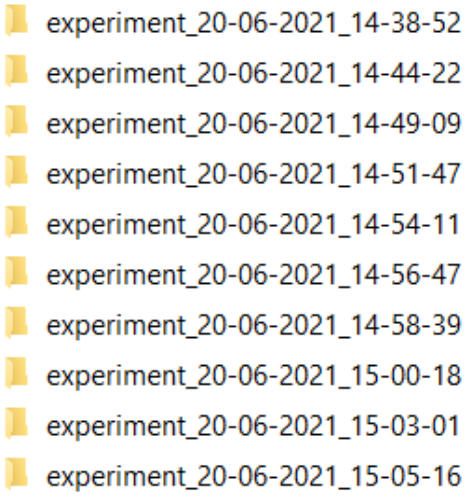
To be able to compare the results and analyse the changes that occur due to these parameters, it is necessary that the objects and the number of knapsacks used in all instances are the same. Thus, it has been established that there will be 1200 objects to be divided into 4 knapsacks. The weights will be 1 for all the objects.

Each time this method is called, a folder containing seven files will be created. This is because we want to see for the same objects and knapsacks how the result changes with six different parameter configurations. Each *experiment_distribution_x* will specify the parameters used for that experiment and the detailed distribution of the objects in the four knapsacks. The fitness, the average expected weight and the weights finally obtained will be indicated in this file too. Moreover, we will also get a summary file. The data used in this study can be found in the reports folders like " experiment_20-06-2021_14-54-11". Therefore, the folder will look as follows:



experiment_20-06-2021_12-32-08 folder

For the analysis of the parameters, the method has been run a total of ten times. Thus, 10 different folders with 7 files each were obtained. From these, conclusions about the different parameters will be drawn. These are the folders used in the study:



Folders generated for the study of parameters

Before showing the data obtained, the chosen parameter configurations will be displayed:

Experiment distribution	Selection amount	Mutation probability	Halting number of generations	Crossover function
0	100	0.01	100	Single point crossover
1	100	0.01	100	Triple point crossover
2	500	0.01	100	Single point crossover
3	100	0.01	1000	Single point crossover
4	100	0.00	100	Single point crossover
5	100	0.20	100	Single point crossover

Therefore, for each report a fitness will be obtained for each configuration. In the following, the data obtained will be shown in 6 tables, each one reflecting a configuration. Report 1 will refer to the oldest one which is the one in the folder "experiment_20-06-2021_14-38-52" and from then on. For each configuration, the average fitness will be calculated and thus it will be possible to see which configuration is more or less optimal. It should be noted that as in our case we are minimizing, the closer the average is to zero, the better it will be.

Experiment distribution 0

Report	Distance to perfection
1	2.0
2	0.0
3	0.0
4	2.0
5	2.0
6	2.0
7	0.0
8	2.0
9	0.0
10	2.0
AVERAGE	**1.2**

Experiment distribution 1

Report	Distance to perfection
1	6.0
2	0.0
3	2.0

4	0.0
5	2.0
6	2.0
7	2.0
8	0.0
9	2.0
10	2.0
AVERAGE	**1.8**

Experiment distribution 2

Report	Distance to perfection
1	2.0
2	2.0
3	0.0
4	0.0
5	2.0
6	0.0
7	0.0
8	2.0
9	0.0
10	0.0
AVERAGE	**0.8**

Experiment distribution 3

Report	Distance to perfection
1	0.0
2	2.0
3	2.0
4	2.0
5	0.0
6	0.0
7	2.0
8	2.0
9	2.0
10	2.0
AVERAGE	**1.4**

Experiment distribution 4

Report	Distance to perfection
1	6.0
2	6.0
3	6.0
4	0.0
5	6.0
6	0.0
7	4.0
8	10.0
9	0.0
10	2.0
AVERAGE	4

Experiment distribution 5

Report	Distance to perfection
1	20.0
2	6.0
3	34.0
4	26.0
5	14.0
6	18.0
7	6.0
8	8.0
9	2.0
10	4.0
AVERAGE	13.8

Analysis of the results

- **Crossover parameter:** for the purpose of reaching conclusions, configurations 0 and 1 have been used, as they have single crossover and triple crossover, respectively. The average obtained from both is very similar, so there is no clear benefit in using one or the other. Even so, by having a slightly better result in the single crossover, using this value for this parameter will probably increase the chances of obtaining a more optimal result.
- **Select amount parameter:** for the purpose of reaching conclusions, configurations 0 and 2 have been used, as they have 100 and 500 respectively in the selection amount parameter. On this occasion the results

have been very similar, although a bit better by putting a larger value on the population. This is since having more population there is more variety to choose from so there may be more optimal chromosomes.

- **Halting number of generations parameter:** for the purpose of reaching conclusions, configurations 0 and 3 have been used, as they have 100 and 1000 respectively in the selection amount parameter. In this case, it was expected that by increasing the halting condition the results would be better, but this has not been the case, and they have even deteriorated a bit. Something better was expected, as more generations would make it more likely to find more suitable chromosomes. It may be that the number of instances is not sufficient to be able to draw a clear conclusion about this parameter.
- **Mutation parameter:** for the purpose of reaching conclusions, configurations 0, 4 and 5 have been used, as they have 0.01,0.00 and 0.20 respectively in the mutation parameter. The first clear conclusion that can be drawn is that a percentage around 0.2 or higher will lead to a considerable deterioration in obtaining a good solution. This is because having a higher probability of mutation, optimal chromosomes can be lost more easily, making it more difficult to obtain a good result. On the other hand, the fact that mutations do not occur also makes the result worse, as these mutations often result in more suitable chromosomes. Therefore, a small mutation probability like 0.01 produces very good results.

Bibliography

First, before starting the assignment, we review unit 5 and practice 4, both of which deal with genetic algorithms:

- https://www.cs.us.es/docencia/aulavirtual/pluginfile.php/10022/mod_resource/content/1/unit-05-2020-21.pdf (Slides from unit 5)
- Code use in practice 4

Once we started to implement both the library for the realisation of genetic algorithm problems and for the modelling of the problem itself, the official Python documentation has been consulted on several occasions as well as other sources:

- https://docs.python.org/3/library/random.html
- https://github.com/microsoft/pylance-release/blob/main/DIAGNOSTIC_SEVERITY_RULES.md#diagnostic-severity-rules
- https://rico-schmidt.name/pymotw-3/collections/namedtuple.html

- https://stackoverflow.com/questions/18296755/python-max-function-using-key-and-lambda-expression
- https://stackoverflow.com/questions/1260792/import-a-file-from-a-subdirectory
- https://stackoverflow.com/questions/4142151/how-to-import-the-class-within-the-same-directory-or-sub-directory
- https://stackoverflow.com/questions/22842289/generate-n-unique-random-numbers-within-a-range
- https://stackoverflow.com/questions/14748910/generate-a-set-of-sorted-random-numbers-from-a-specific-range
- https://stackoverflow.com/questions/58792963/time-complexity-for-adding-elements-to-list-vs-set-in-python
- https://stackoverflow.com/questions/53422887/python-how-do-i-randomly-mix-two-lists/53423019
- https://stackoverflow.com/questions/7529376/pythonic-way-to-mix-two-lists
- https://stackoverflow.com/questions/47406741/disable-auto-wrap-long-line-in-visual-studio-code
- https://stackoverflow.com/questions/63314452/python-autopep8-formatting-not-working-with-max-line-length-parameter
- https://softwareengineering.stackexchange.com/questions/308972/python-file-naming-convention
- https://stackoverflow.com/questions/9195455/how-to-document-a-method-with-parameters
- https://stackoverflow.com/questions/58622/how-to-document-python-code-using-doxygen