**METAL CUTTING COURSEWORK**

## 1. Explanation and justification

## 1.1 Items represented as groups

The chromosome is represented as an array of different groups where each group (gene) consists of pieces that are to be cut from the same stock. To calculate the fitness of a solution, the evaluation function will assign the best stock size to each group. The stock size must not be less than the sum of all the pieces in the group and has the lowest cost. The total cost of the solution is then calculated by summing up prices of the assigned stocks.

Figure 1 illustrates the mapping.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 67, 33 | 44, 56 | 32, 39, 32 | 59, 48 | 63, 33, 22 | 44, 56 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 100 | 100 | 105 | 110 | 120 | 100 |

**genotype**

item chromosome

best fit stock size chromosome

**phenotype (solution)**

Activities

Stock size

67, 33

100

43, 55

100

32, 39, 31

105

59, 48

110

63, 34, 22

120

the cost for this solution: 10 + 10 + 10.5 + 11 + 12 +10 = 63.5

100

44, 56

Figure 1 Symbolic representation of mapping

## 1.2 Important parts of the algorithms

## Encoding

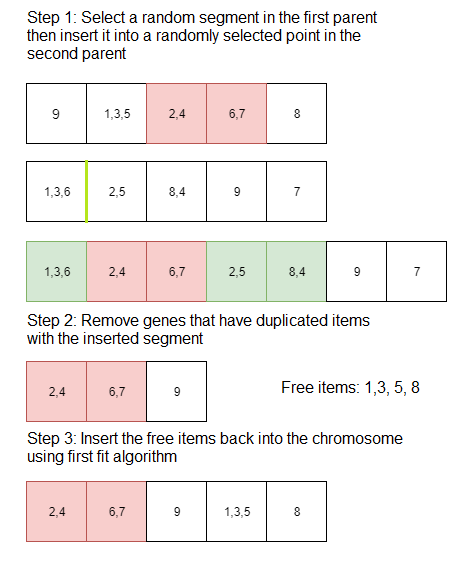
## 

## A custom first fit algorithm is used to generate the initial random population and allow crossover and mutation operators to build new off springs.

## Crossover

## The crossover operator is a modified version of Falkenauer's Bin Packing Crossover (BPCX) [1].

## In order to create an offspring, a segment of a parent is randomly selected and insert into a random point in a copy of the other parent's chromosome. Groups that have pieces that are duplicated in the inserted part are removed. The removal of groups will cause some pieces that are not duplicated in the inserted part to be missing from the chromosome. These pieces are then to be regrouped into the chromosome by using the first fit function.



## Mutation

## The mutation operator makes a copy of the parent's chromosome. It then randomly selects groups (genes) from this copy to remove. The removed pieces are then regrouped into the chromosome by using the first fit function.

## 

## C:\Users\Thai Con\AppData\Local\Microsoft\Windows\INetCache\Content.Word\mutation (2).png

## 1.3 Genetic algorithm and Artificial immune system

## Genetic algorithm

## I use tournament selection method for my genetic algorithm along with the two reproduction operators explained above.

## Artificial immune system

## The mutation operator is similar to the one above with one small modification. Instead of picking groups for removal randomly, I used the formula discussed in the lectures to calculate how many groups are to be removed. Removed pieces are then regrouped by using the first fit algorithm just like the mutation operator for the genetic algorithm

## 2. Novelty and insight

## Recursive Revised First Fit Algorithm (RRFF) [2]

## First fit algorithms are design for single stock size problems. For our problem, I used RRFF, a modified version of the first fit algorithm that was proposed by K.Takeyasu [3]. In order for this function to perform well on our problem, I calculate the yield rate differently from the paper.

## Yield rate = volume used / stock price

## By calculating the yield rate like this, our algorithms will be better at pick the right stock size.

## Sort items before applying first fit function

## When we use the first fit algorithm to either initialise the population or to regroup the free items as part of the reproduction operators, I found that sorting them in an increasing order would help our computational intelligence algorithms find solution with lower cost.

## Index map

## In order for the reproduction operators to work, the chromosomes must not contain duplicate items. To overcome this problem, I created a piece\_map which is essentially an array that contains the indexes of the pieces.

## The chromosome instead of storing pieces, it stores the indexes of these pieces.

## For example, with the first data set, I store the pieces in a list like this:

PIECE1 = [3, 3, 3, 3, 3, 4, 4, 5, 6, 6, 7, 7, 7, 7, 8, 8, 9, 10, 10, 10]

## The corresponding piece map for this would be:

PIECE\_MAP = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

## Tournament selection

## Since our problem is a minimisation problem, I decided to use tournament selection over roulette wheel selection. One reason for this was that, to calculate the probability for the roulette wheel, I had to take the inverse of the cost of solution which causes division by zero error when I tried to add all the probabilities cumulatively.

## Generational model for GA

## By observing the results, I found that replacing the entire population with new generation at the end of iteration gave me better solutions.

## 

## 3. Comparing algorithms

## My observation after testing out the algorithms were that artificial immune system (AIS) finds better solution than genetic algorithm (GA) when the number of function evaluation is small. On the other hand, with a large number of function evaluation, genetic algorithm outperform the artificial immune system.

## I set up two hypothesis tests.

## I ran the two algorithms with 500 function evaluation 30 times. The solutions found by AIS has an average cost of 1765.567 and 1767.592 is the average cost of solutions found by GA. I then perform Welch's test on the data set and get a p-value of 5.51E-35 which is less than 0.05. So my first claim was correct.

## I then ran the two algorithms with 2000 function evaluation 30 times. The solutions found by AIS has an average cost of 1765.433 and 1767.633 is the average cost of solutions found by GA. Performing Welch's test on the data set and get a p-value of 0.043691. So the second claim is rejected because the average cost found by AIS is less than the average cost found by GA.

## In conclusion, AIS seems to find better solution compared to GA's given the same number of function evaluation. Although the differences gets smaller as the number of function evaluation increases.

References

[1] Falkenauer, E. and Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing. *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pp.1186-1193.

[2] Hinterding, R. and Juliff, K. (1993). A Genetic Algorithm for Stock Cutting: An Exploration of Mapping Schemes. *Technical Report*.

[3] Toyoda, J. and Takeyasu, K. (2008). A Recursive Revised First Fit Algorithm for Cutting Stock Problem. *International Journal of Information Systems for Logistics and Management*, 4(1), pp.31-40.