Comp 304 Assignment 2

Genetic Algorithms

217008024

Table of Contents

Introduction 2

Chromosome and initial population and population size used 2

Fitness function used 3

Selection method used 4

Mutation Operator and mutation rate 5

Crossover operator and crossover rate 5

Termination criteria 6

Introduction

Genetic algorithms mimic human evolution and can be used to solve a number of problems, provided that they are structured properly.

In this delivery problem, we have a delivery truck that needs to deliver items (boxes). The truck can hold a certain maximum weight, *capacity*, which is taken from the test data, *input.txt*. The problem then is to determine which items can be placed into the truck.

This is done by using the comparative values of these boxes to determine a sort of minimum value, *quota*, eligible for delivery, which is also taken from the test data, *input.txt*.

The genetic algorithm then has to determine which boxes contain the highest combined value to be eligible for delivery, whilst still being under the weight constraint.

Chromosome and initial population and population size used

The chromosomes are created by the chromosome class. They are objects with constructors to hold their genes. These genes are of type ArrayList<Integer> and contain binary bits 0,1. This is to represent the decision of whether we want to put the item into the vehicle (represented by 1 bit) or whether we don't want to put the item into the vehicle (represented by 0 bit).

The initial population is taken directly from the input file as described in the Population class. This class then reads the input file further to determine whether or not the objects in the population comply to the regulations set out in the assignment documentation (i.e. no population should contain more than 20 objects).

The population size is then expanded when the genetic algorithms are applied and are in effect. This population is grown to a size of 3000 and is suitably maintained by the program. I have found that higher values severely impact system performance and optimizing for this value yields the same accurate results albeit at a lesser cost of system resources.

This population size is taken into account when population breeding is in effect and as such, members of the population that exceed this imposed limit of 3000 are deemed unfit for breeding/selection and are removed from consideration. Members are sorted and therefore we can rest assured that the fitter members are toward the top of this population list and exempt from immediate exclusion.

Fitness function used

The fitness is evaluated at two points, the calculation of fitness for a single member/chromosome, and the calculation of fitness for the population/or a set of members that may result in a solution/population.

In both cases the fitness calculation method is identical.

We have to decide whether we to include the item in the truck or not include the item in the truck.

For this we use the fitness calculation for a single member. The calculation for this is determined by the gene list in the chromosome class. For each selected member, their gene list either contains 0 or 1 – where 0 = not included and 1 = included.

Should the member's gene return a 1, it is then considered for evaluation by the fitness function.

The fitness function then adds the members corresponding weight and value to that of the selectedMembers to make up the weight and value variables. After this allocation is complete the function then compares the weight with the capacity of the truck.

If the weight of this single item exceeds the capacity of the truck, it is instantly given a fitness value of 0 – deeming it unfit. This ensures that the capacity limit is not exceeded.

Should it pass this condition and come in under the truck's capacity, its weight gets added to the totalSelectedMembers and its fitness is assigned the value of its own value (I.e. fitness = item value).

This is done so that the calculation is simplified whist still being accurate as time progresses over the evolutionary process.

Similarly, the fitness function to calculate the fitness of the new population/solution set takes the totalWeight of the totalSelectedMembers thus far, and compares them to the capacity of the truck. If a member is added to this totalSelectedMembers list that happens to break/exceed this capacity limit, that member is given a fitness of 0 and is deemed not suitable – this will be addressed in further methods where this member of fitness 0 will be eliminated from the population.

Should all conditions pass however, the member is accepted into this totalSelectedMembers and its value is added to the fitness of this list.

Selection method used

Finding a selection method was challenging as there were two good options for this problem, the roulette selection and the tournament selection. I have opted for the tournament selection here.

Under the tournament selection, 2 chromosomes in the population are chosen at random and they compete with their respective fitness values – the chromosome with the higher fitness value wins and moves on to the next round.

This method is not fool-proof as selection size can get quite large and unnecessarily use up resources on some chromosomes with lower fitness or by continuing the tournament selection when an optimal solution has already been found. For this reason, unfit members are removed from the population, thereby allowing the selection size to be a more tight-knit selection process.

The members are chosen and sorted into a list such that the highest fitness members are at the very top (i.e at positions 0, 1, 2,… and so on). This *sort* function does the work of the tournament for us and selected members are then chosen to be added back into the population, in order of their fitness.

Mutation Operator and mutation rate

Just like in evolutionary theory, mutation is the manipulation of ones DNA to fundamentally adapt the being so that it may survive based on new environmental/survival conditions.

The Mutate Operator is called from within the breedPopulation method in the GeneticAlgorithm class. The mutate operator is called on a child chromosome and executes a change in the DNA of that chromosome. This is done by iterating through each gene of the chromosome and flipping a bit value, either from 0 to 1 or 1 to 0 and is accomplished using the Mutation_Rate variable.

The Mutation_Rate is set to 0,025 in the GeneticAglorithm class. The function receives this mutation rate and goes about analyzing the genes for the chromosome it chooses to mutate. The ThreadLocalRandom class provides us a random selection of 0s and 1s which we then check against our murationRate. Should the mutationRate be less than the random selection, the chromosome remains as is, however should the mutationRate be greater than the random value, the chromosome is mutated at that iteration.

Mutation rate has to be low as there would be too much variance in the population if we allowed for rapid mutation and a solution may take longer to find.

Crossover operator and crossover rate

The crossover operator is used to create child chromosomes from parent chromosomes. These child chromosomes may then be manipulated and have their characteristics changed just as in human evolution.

The Crossover Operator is called from within the breedPopulation method in the GeneticAlgorithm class. In this class 2 parent chromosomes pass their genes through to be crossed over. Crossover operator selects random bits at random from both parent chromosomes to be placed within the child chromosome. This is done by selecting a portion of parent chromosome 1 and a portion of parent chromosome 2, and placing the respective pieces inside the child chromosome – thereby giving it a DNA consisting of a crossover of their two chromosomes. This is done entirely at random using the ThreadLoacalRandom class and therefore there is no set rate of crossover.

Termination criteria

The termination criterion for this program is when the end of the file is reached or when the input data contains more than the allotted 20 objects in a population as described in the assignment documentation.