**INFO6205 37198**

**Program Structure & Algorithms SEC 05 - Spring 2018**

**Team Project - Genetic Algorithms**



|  |  |
| --- | --- |
| Team member name | Zhiyong Zhang |
| Team member name | Yixuan LI |
| NUID | 001833676 |
| NUID | 001822594 |
| Team Number | #516 |

Github link:

contents

[1. Requirement 3](#_Toc511310983)

[1.1 basic components 3](#_Toc511310984)

[1.2 basic concepts 3](#_Toc511310985)

[2. Project Objectives 4](#_Toc511310986)

[3. Runtime Environment 4](#_Toc511310987)

[4. System Design 4](#_Toc511310988)

[4.1 constants 4](#_Toc511310989)

[4.2 parameters 5](#_Toc511310990)

[5. Control Flow 5](#_Toc511310991)

[6. Code Analysis 5](#_Toc511310992)

[6.1 data structures 5](#_Toc511310993)

[6.2 functions 5](#_Toc511310994)

[7. Result Analysis 5](#_Toc511310995)

[7.1 chart 5](#_Toc511310996)

[7.2 analysis 5](#_Toc511310997)

[8. Test 5](#_Toc511310998)

# Requirement

## 1.1 basic components

1. a genetic code (or use the four bases of DNA for simplicity) and a random generator/mutator of such codes;
2. gene expression: how do individual genes code for particular traits--a symbol table using a hash function?
3. a fitness function--this is essentially a measure of how good a candidate (organism) solution is for the problem you have chosen to solve;
4. a sort function (priority queue is best) -- to order the organisms by their fitness function;
5. an evolution mechanism--this takes care of the seeding of generation 0, and the births and deaths between generation N and N+1;
6. a logging function to keep track of the progress of the evolution, including the best candidate from the final generation;
7. a set of unit tests which ensure that the various components are operating properly;
8. (optional) a parallel computation mechanism so that you can divide your population up into colonies (sub-populations) and create the next generations for each colony in parallel;
9. (optional) a user interface to show the progress of the evolution.

## 1.2 basic concepts

1.2.1 modes of reproduction

Asexual: from a single parent

Sexual: from two parents

1.2.2 source of randomization

Crossover: some of the genes of the offspring come from the mother and some from the father. It is the main source of randomization in sexual reproduction.

Mutation: It is relatively rare. Nevertheless, mutation is the only mechanism available to asexual organisms.

1.2.3 genotype and phenotype

Genotype: The set of the genes in an organism is called the genotype. In nature, genes are coded in DNA。

Phenotype: Phenotype has fitness in an environment. It is made up of a set of individual traits, each of which corresponds to a single gene. The phenotype is expressed from the genotype.

# Project Objectives

The goal of the project is to develop a genetic algorithm and to use it to find a good solution to a highly complex problem.

The basic idea behind GAs is to model the search for a solution in the same way that organisms in nature are adapted to their environment and lifestyle. You start with a random population (generation 0) and you "breed" successive generations from that. In each generation you "cull" those organisms which are unfit for the environment and you give birth to new organisms which are based, genetically, on the organisms which are fit.

# Runtime Environment

|  |  |
| --- | --- |
| Operating System | Mac OS |
| Model Name | MacBook Pro |
| Processor Name | Intel Core i5 |
| Processor Speed | 3.1 GHz |
| Number of Processors | 1 |
| Total Number of Cores | 2 |
| Memory | 8 GB |

# System Design

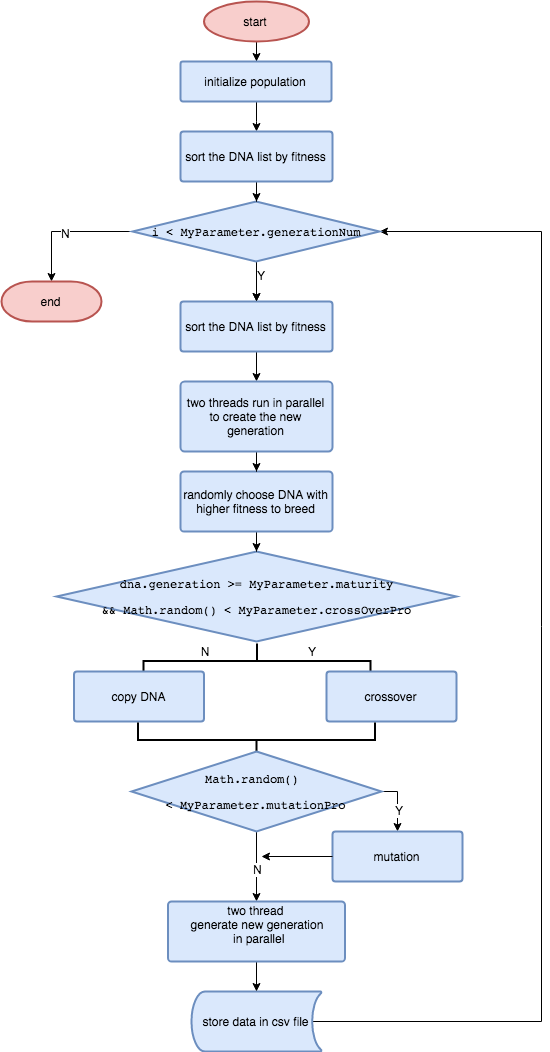
## 4.1 constants

|  |  |  |
| --- | --- | --- |
| variable | value | description |
| itemNum | 2000 |  |
| weightMax | 10000 |  |
| coreNum | 2 |  |

## 4.2 parameters

|  |  |  |
| --- | --- | --- |
| variable | value | description |
| populationNum | 1000 |  |
| survivePro | 0.5f |  |
| fecundity | 2 |  |
| maturity | 1 |  |
| generationNum | 10000 |  |
| crossOverPro | 0.5f |  |
| mutationPro | 0.01f |  |

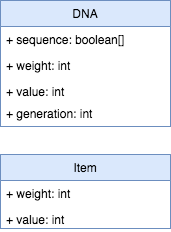
# Control Flow



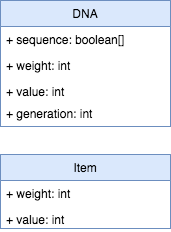
# Code Analysis

## 6.1 data structures

6.1.1 DNA



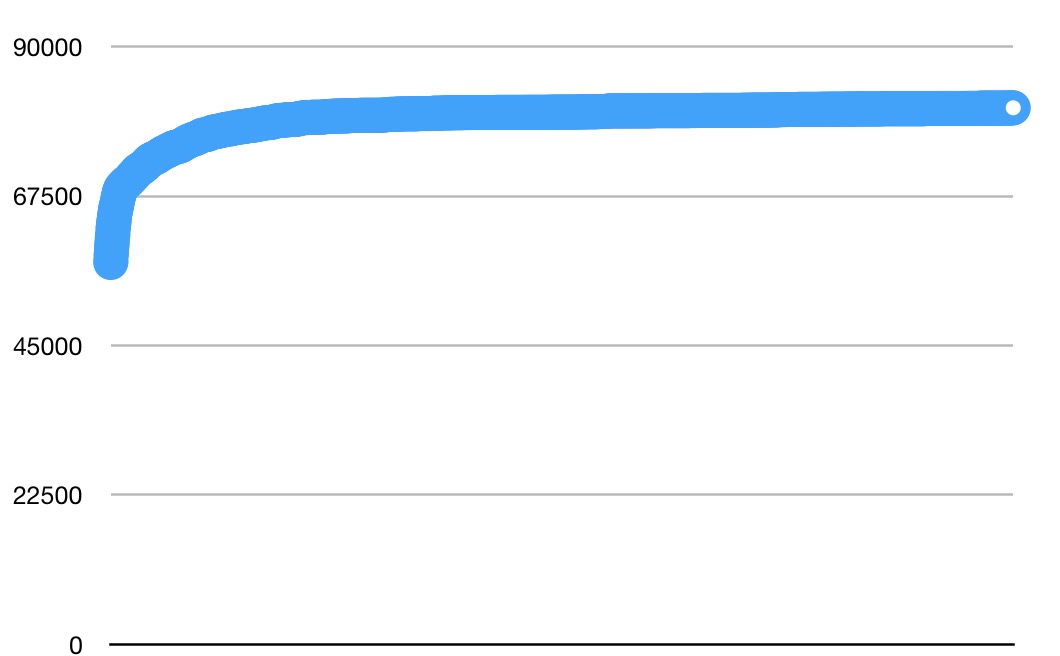
6.1.2 Item



## 6.2 functions

# Result Analysis

## 7.1 chart



## 7.2 analysis

# Test

start with a random population (generation 0) and "breed" successive generations from that.

In each generation "cull" those organisms which are unfit for the environment and give birth to new organisms which are based, genetically, on the organisms which are fit.

I recommend that you model only *asexual*reproduction (no crossover, only mutation). But I know that many of you will want to use crossover which results (typically) in faster evolution. However, sexual reproduction (using crossover) is *more complicated.*For any given problem, there will only be one environment.

I can't tell you in advance how many generations you will need to run, but something like 100 will probably be good enough. Be careful that you have a well-defined terminating condition.

sing JUnit as your test runner. Try to put all of your assumptions (those "magic" values in the code) in one central place thus making it easier to adjust parameters when you run an evolution.

For example, the proportion of organisms that must be culled due to lack of fitness, or the proportion of organisms that will "bud" a daughter organism each generation, etc.

There should be one main program which will run, and which will run the evolution and announce the best solution. There is no UI-component as such (unless you are a three-person team and would like to choose that option).

I recommend that you have one team member concentrate on the genetic/fitness aspects of the work, while the other concentrates on the evolution/logging aspects.