GA Implementation of Auto-Composing Test Paper

Xiaoge Hou , Junhan Ma

1. Problem Description

Our project plans to implement Genetic Algorithm on composing exam paper automatically based on problem database. For instance, figure out an efficient way to extract total 40 problems from database whose distribution is 10 multiple choices, 10 true or false, 10 blanks and 10 calculations, additionally its total score is 100 and expecting 0.72 difficulty and 0.8 knowledge points coverage.

2. Algorithm Design [1]

Mapping basic genetic algorithm conceptions into our practical problems, we designed a series of classes to implement and solve the problem.

2.1 Gene code: Problem class

In our problem, the problem id would be the gene concept in genetic algorithm because a list of different problems consists of a paper and decides the paper's traits. In our project, the problem entity has fields of id, type, difficulty, score and a set of knowledge points id.

2.2 Gene data bank: ProblemDB class [2]

We designed a problem database to store total 5000 gene code. In another word, we hardcode 5000 different problems in database.
a. Type 1: multiple choice for 1 score from id 1 to 1000;
b. Type 2: multiple choice for 2 score from id 1001 to 2000;
c. Type 3: true or false for 2 score from id 2001 to 3000;
d. Type 4: blank completion whose score is 1 to 4 randomly from id 3001 to 4000;
e. Type 5: question and answer whose score is difficulty*10 from id 4001 to 5000;
All problems' difficulty is randomly generated from 0.3 to 1 and knowledge points count is from 1 to 4, points id is from 1 to 100 (100 kinds of points in total id from 1 to 100).

2.3 Gene expression: Paper class

Paper is the unit which is composed by a set of problem(Genes). Its traits include id, total score, difficulty, KPCoverage(knowledge point coverage) and adaptation degree(fitness).

Fitness function in paper class is defined by difficulty and KPCoverage [2][4].

$$f = 1 - \left(1 - \frac{\mathrm{M}}{\mathrm{N}}\right) \times f1 - |EP - P| \times f2$$

* M/N is KPCoverage, EP is difficulty expectation, P is unit difficulty;
* f1 is KPCoverage weight, f2 is difficulty weight, in our project f1 is 0.2 and f2 is 0.8 who are final parameters defined in GlobalWeight class.

2.4 Population: Population class

Population class is the generation of papers, a list of papers. The getBestFitnessPaper() function is going to sort papers by fitness and get the best fitness candidate. Which would be used in selection process in evolution.
The population would be generated by the given rule including constraints of each problem count and total score.

2.5 Unit standards: Rule class
The rule class defined constraints of total score and each problem type count for generating a paper. And also, the expectation for ideal paper like difficulty and a set of knowledge points hope to cover.

2.6 Evolution mechanism: GA class
Flow of Genetic Algorithm for auto-composing test paper:

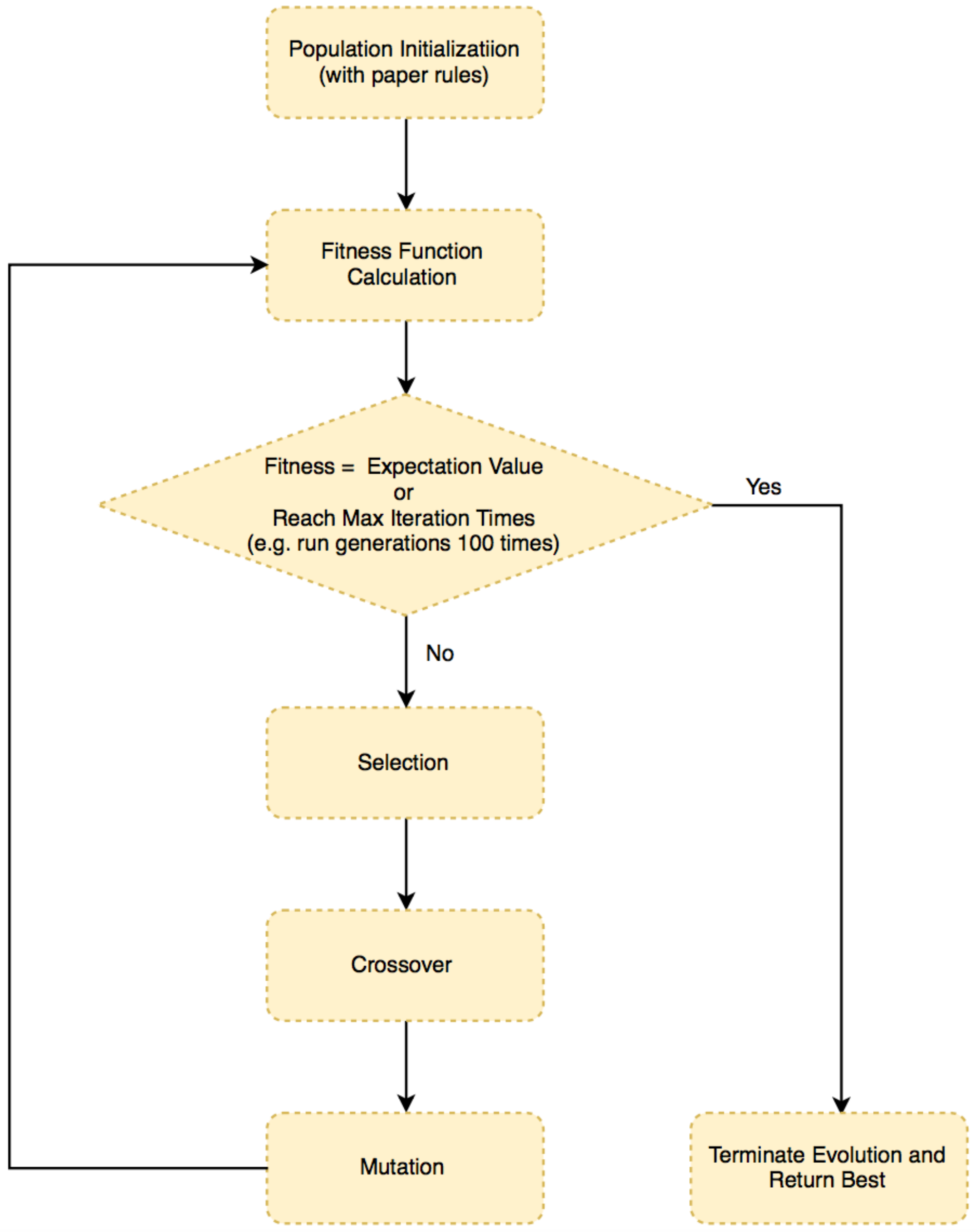

Population Initializatiion (with paper rules)
Fitness Function Calculation
Fitness = Expectation Value or Reach Max Iteration Times (e.g. run generations 100 times)
Yes
No
Selection
Crossover
Mutation
Terminate Evolution and Return Best

Figure 1. Genetic Algorithm Flow Chart

Evolution methodology: In order to prevent deviation of the evolutionary direction, which is easy to cause local optimal solution, elitism evolution method is used in this project, which means the current fittest member of the population is always propagated to the next generation.

Selection function: Get some good individuals as parents to generate the next generation. In specifically, randomly generate a tournament array, select the best fitness paper in the tournament array as the result of selection, which is the process of simulating the survival of the fittest.

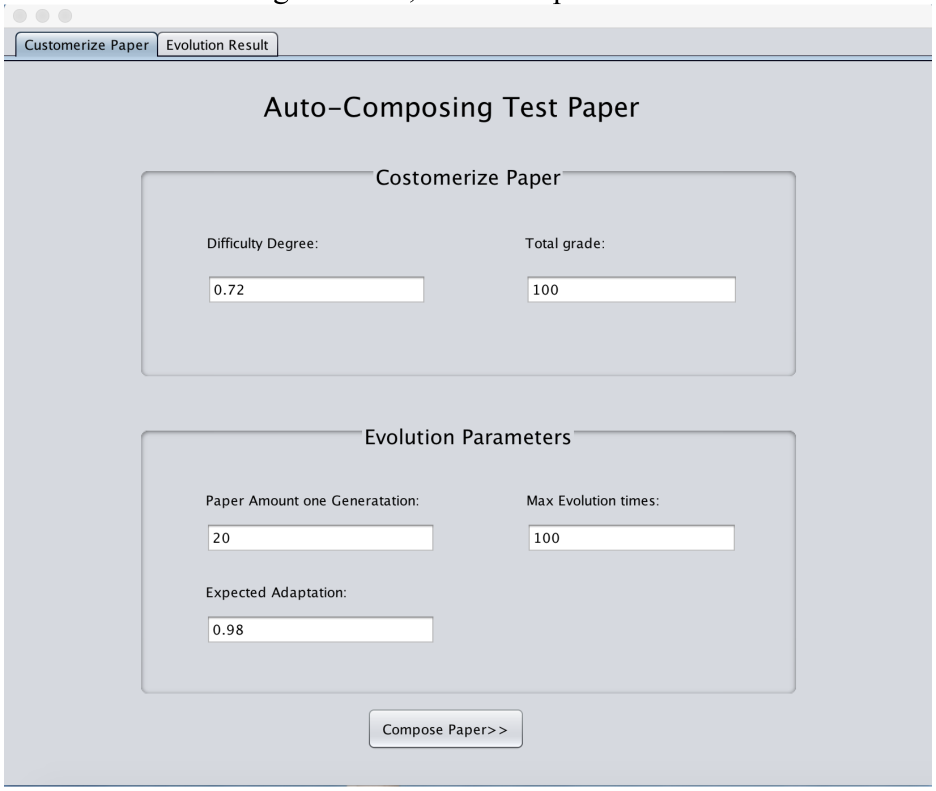
Crossover function: Crossover is a reorganization of chromosomes. We use multi points crossover strategy: randomly generate two integers n1, n2 between (0, N), where N is the number of genes of one chromosome, and gene segment from two parents between n1 and n2 or others except segment between n1 and n2 are reorganized to get a new off-spring. But must ensure that the genes are not duplicated.

Mutation function: Each individual gene has a chance to mutate. If the random probability is less than the mutation probability which is 0.002 in this project, the gene can be mutated. The principle of the mutated gene is that it has the same problem type, the same score, and the same knowledge points as the original gene.

Terminate condition: When there has been no improvement in the population for 100 iterations or the fitness has reached an expectation value, we terminate the evolution and get the best paper.

3. Result

We designed UI to display paper composing process in MainJFrame. Figure 2 is asking for rule of difficulty degree and total grade, additionally asking for parameters of evolution constraints of paper amount in one generation, max evolution times and expected adaptation. Figure 3 is the composing result showed us first generation, evolution process and final result.



Auto-Composing Test Paper

Costomerize Paper

Difficulty Degree: 0.72

Total grade: 100

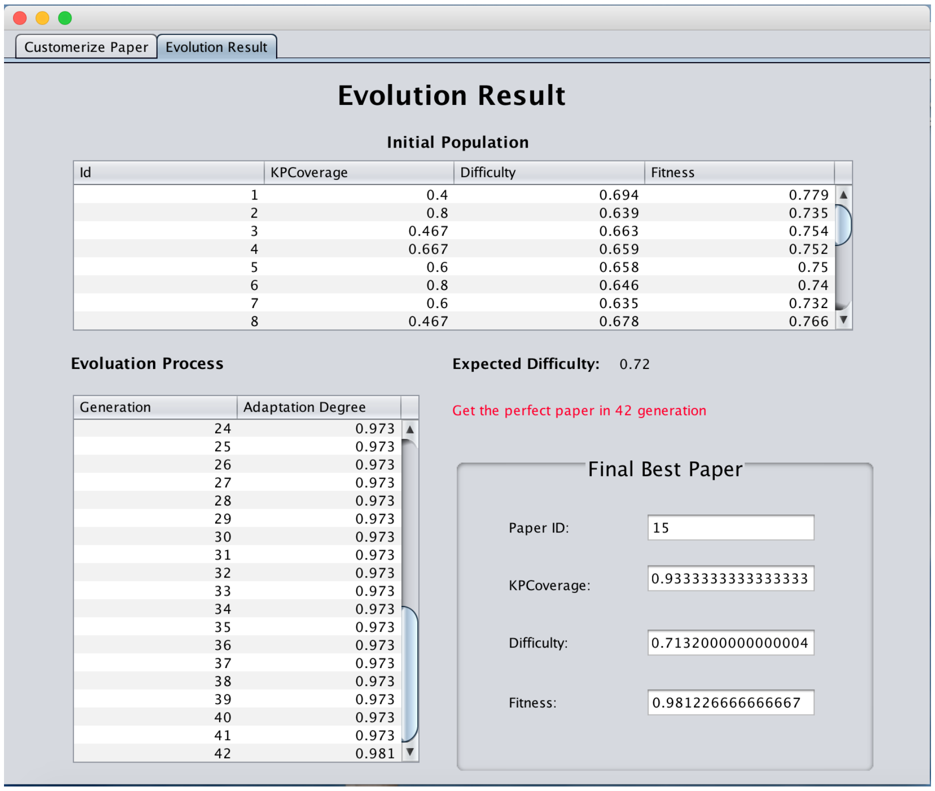
Evolution Parameters

Paper Amount one Generatation: 20

Max Evolution times: 100

Expected Adaptation: 0.98

Figure 2. Settings



Evolution Result

Initial Population

Id	KPCoverage	Difficulty	Fitness
1	0.4	0.694	0.779
2	0.8	0.639	0.735
3	0.467	0.663	0.754
4	0.667	0.659	0.752
5	0.6	0.658	0.75
6	0.8	0.646	0.74
7	0.6	0.635	0.732
8	0.467	0.678	0.766

Evoluation Process

Generation	Adaptation Degree
24	0.973
25	0.973
26	0.973
27	0.973
28	0.973
29	0.973
30	0.973
31	0.973
32	0.973
33	0.973
34	0.973
35	0.973
36	0.973
37	0.973
38	0.973
39	0.973
40	0.973
41	0.973
42	0.981

Expected Difficulty: 0.72

Get the perfect paper in 42 generation

Final Best Paper

Paper ID:	15
KPCoverage:	0.9333333333333333
Difficulty:	0.7132000000000004
Fitness:	0.981226666666667

Figure 3. Result

(Note: Sometimes we cannot get the ideal paper satisfied all constraints which can be solved by lower expected adaptation or increase max evolution iterations.)

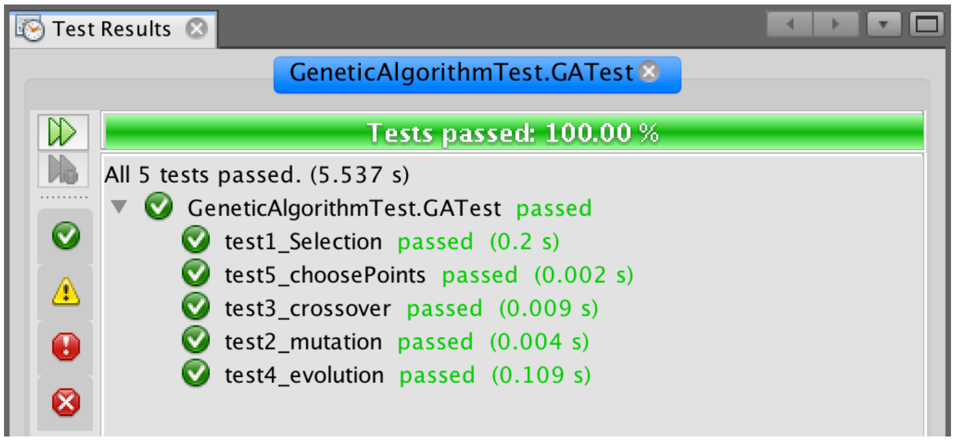
4. Unit Tests

4.1 Unit Tests Design

We designed unit tests for important functions in projects including followings:

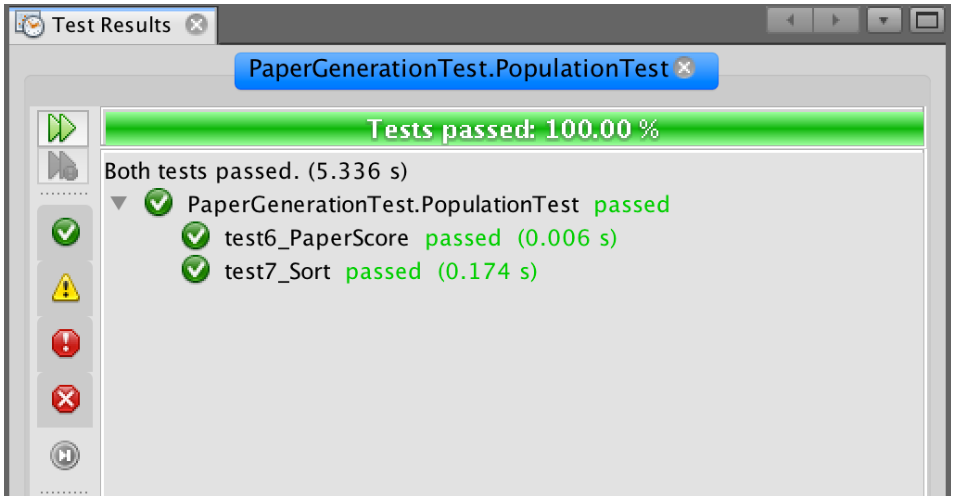
a. Population class constructor to generate correct population;
b. Get best fitness paper function (mainly sort by fitness function);
c. Selection function;
d. Mutation function;
e. Crossover function;
f. Evolution function;
g. Choose multi points function in crossover process

4.2 Unit Tests Result



```
Tests passed: 100.00 %
All 5 tests passed. (5.537 s)
GeneticAlgorithmTest.GATest passed
    test1_Selection passed (0.2 s)
    test5_choosePoints passed (0.002 s)
    test3_crossover passed (0.009 s)
    test2_mutation passed (0.004 s)
    test4_evolution passed (0.109 s)
```

Figure 4. Test Result-part1



```
Tests passed: 100.00 %
Both tests passed. (5.336 s)
PaperGenerationTest.PopulationTest passed
    test6_PaperScore passed (0.006 s)
    test7_Sort passed (0.174 s)
```

Figure 5. Test Result-part2

5. Reference

[1] https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
[2] http://www.cnblogs.com/artwl/archive/2011/05/20/2052262.html
[3] https://www.jianshu.com/p/7fe9d3bb00ac
[4] https://github.com/jslixiaolin/GADemo
[5] https://ieeexplore.ieee.org/document/5982470/?reload=true&arnumber=5982470