Department of Industrial Engineering and Management,

Yuan-Ze University, Taiwan, R.O.C.

The Production Scheduling and Soft-Computing Lab

The Genetic Algorithm for Solving the Quadratic Assignment Problem and Continuous Problem

2005/5/12

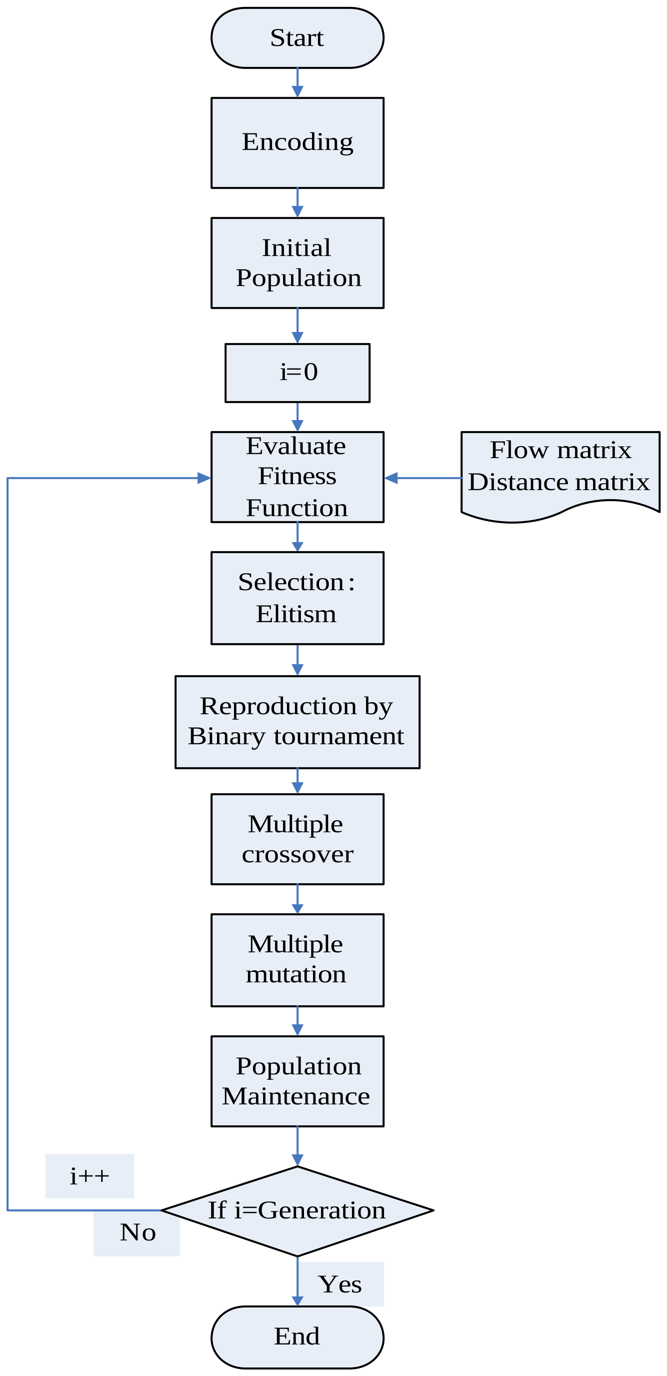
1. Introduction

The work solves the quadratic assignment problem (QAP) and continuous problem by genetic algorithm (GA). Then, we also introduce some techniques that attend to improve the solution quality, including the elitism, multiple crossover operators, and multiple mutation operators. Furthermore, the work also develops a callable component, which is named OpenGA. The goal of designing the component is to reduce the complexity and easy to use when we solve different kinds of problem. The work is organized as following.

The section 2 and section 3 describe how to solve QAP and continuous problem by GA. Then, the study composes an object-oriented component written in Java. The study describes the structure of the OpenGA by UML diagrams, which presents in section 4. Section 5 is the experimental result of the two cases and the section 6 is the discussion and conclusions.

2. Solving the QAP

The study presents the fundamental procedures and methods of GA to solve the QAP. Therefore, the encoding method and generating an initial solution are discussed in the beginning. Then, the next ones are evaluating the new solutions, fitness assignment, selection and elitism, crossover, mutation, replacement, and stopping criterion. Furthermore, the elitism strategy, multiple crossover operators, and multiple crossover operators are included in the procedure. The figure below expresses the procedures of the modified GA. The following sub-sections describe them in detail.



2.1 Generate an Initial Solution

The encoding strategy for the QAP, which is one of the combinatorial optimization problems, the sequential encoding type is employed here. Then, each chromosome can be randomly generated an initial solution for the problem which assigns each department at exactly one dimension. The work adopts the Nug30 as an example and the figure 2.1 presents the encoding of QAP for the 30 departments, which shows the department 14 at the first position, department 5 at the second position, and so on. Thus, the initial solution is done after we deal with the encoding and to generate initial solution.

14	**5**	**28**	**24**	**….**	**12**	**11**	**23**

Figure 2.1 the problem representation of the QAP for 30 departments

2.2 Evaluate the Objective Value

The objective function of QAP is

$$min\, Z = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} \mathrm{f}_{ij} \times \mathrm{D}_{ij} \tag{1}$$

D_{ij} : The distance between the locations at *i* and *j*.

f_{ij} : The flow between the departments *i* and *j*.

By the objective function, it calculates the summary distance from one department to the other one by the flow quantity between the two departments. Suppose the new solution is 14-15-28…-23, the procedure to perform the calculation of the objective value are:

For department 14: 4*5 + 5*1 + 6*5 + … + 6*5 + 5*0 + 5*1
For department 5 : 4*2 + 5*0 + … + 3*0 + 2*6 + 4*4
For department 28: 4*0 + … + 6*0 + 5*3 + 3*1
⋮ ⋮
⋮ ⋮
For department 12: 0*0 + 2*6
For department 11: 3*0

The optimal assignment of Nug30 in QAPLIB is shown as 14, 5, 28, 24, 1, 3, 16, 15, 10, 9, 21, 2, 4, 29, 25, 22, 13, 26, 17, 30, 6, 20, 19, 8, 18, 7, 27, 12, 11, and 23. Besides, its objective value is 3062.

2.3 Fitness Assignment

The fitness value is depended on the objective value and it is to determine the goodness of a chromosome. The work uses the normalization method to transform the fitness value. The equation is as follows.

$$fitness_i = \frac{f_i - f_{\min}}{f_{\max} - f_{\min}} \qquad (2)$$

where

f_i : The objective value of individual *i*.
$f_{\min}$: The minimum objective value of current population.
$f_{\max}$: The maximum objective value of current population.

After the normalization, each fitness value is between 0 and 1 and the larger objective value yields larger fitness value. Because the problem is a minimization problem, we select the chromosomes with smaller fitness value.

2.4 Elitism and Selection

The purpose of elitism is to preserve better chromosomes so that crossover operator or mutation operator won' t destroy it, and keep it to the next generation. If it's a single objective problem, we use selection sort algorithm which is a type of sorting algorithm. Therefore, we select a proportional better chromosome and store them into the external achieve. For example, if the elitism is 20% of original population size, we pick the top 20% individual. Then, if it's multiple-objective problem, we select all non-dominated solutions into archive.

Besides, the better individuals in the archive will be selected into the mating pool. The size of selecting from archive is depended on the proportion we set before. If it's 20%, we select 20%*popSize of better chromosomes into the pool. When the 20%*popSize is less than the actual length of archive, we directly copy them all into the mating pool. However, if the length of archive is larger than the 20% of original population size, we randomly pick the 20% of individuals from the archive.

After selecting the elite chromosomes, the next stage is accomplished by binary tournament. We randomly pick two individuals in the same population and compare their fitness value. The chromosome with smaller fitness value will be the winner and put it into the mating pool. The procedure is repeated until the size in the mating pool is up to the population size.

2.5 Crossover

The selection procedure selects better chromosomes into the mating pool. The crossover procedure is randomly selecting two chromosomes to mate. There are several crossover methods for combination problem. Besides, the multiple crossover strategy is used in the GA procedure. They are the position-based crossover and PMX (Partial Message Crossover). They are shown as follows:

Syswerdra (1989) proposes a position-based crossover. There are two versions of the method, including two point crossover and single point crossover. We demonstrate the former one by an example. The steps are:

1. Select two chromosomes and named it as parent 1 and parent 2.
2. Determine the two cut points, suppose they are at i and j, transfer the genes which outside the range from i to j to the offspring in the same position. From the figure 2.2, the cut points are at 3 and 7 and the transferring result is at figure 2.3.
3. Copy othe r genes which inside the range of parent 1 in the order of relative gene position of parent 2 and it shows at figure 2.4.

			↓				↓			
Position	1	2	3	4	5	6	7	8	9	10
Parent 1	11	12	13	14	15	16	17	18	19	20

Position	1	2	3	4	5	6	7	8	9	10
Parent 2	11	16	14	17	12	13	19	20	18	15

Figure 2.2 The two parent chromosomes and determination of the two cut points

Position	1	2	3	4	5	6	7	8	9	10
Offspring	11	12						18	19	20

Figure 2.3 Copy genes of Parent 1

Position	1	2	3	4	5	6	7	8	9	10
Offspring	11	12	16	14	17	13	15	18	19	20

Figure 2.4 Copy other genes to offspring.

The PMX process is as follows：

Step 1: Select two parents P_1 and P_2 randomly from population.
Step 2: Generate two crossover points cp_1 and cp_2 randomly. Then, exchange two substrings, which defined by two positions, between parents P_1 (4, 5, 6, 7 in Figure 2) and P_2 (2, 8, 3, 4 in Figure 2) to product C_1 and C_2.
Step 3: Exchange the genes which are already in the substring from the parent P_2 and P_1, for example the second gene of C_1: 2->4->7.

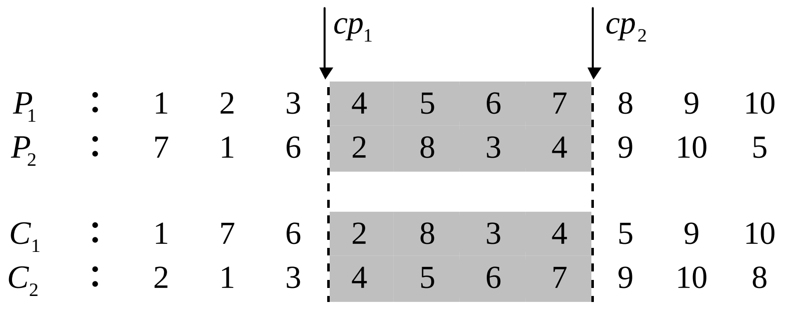

cp_1
cp_2
P_1 : 1 2 3 4 5 6 7 8 9 10
P_2 : 7 1 6 2 8 3 4 9 10 5
C_1 : 1 7 6 2 8 3 4 5 9 10
C_2 : 2 1 3 4 5 6 7 9 10 8

Figure 2.5 PMX process

2.6 Mutation

The purpose of mutation like crossover is to do a variation on the current chromosome. The same with the crossover stage, we also employ the multiple mutation strategy. The mutation strategies include swap mutation, and shift mutation. They are described below.

The swap mutation is very easy to implement because it just has to set two positions and exchange the two values of its position. The result is shown in the figure 2.6.

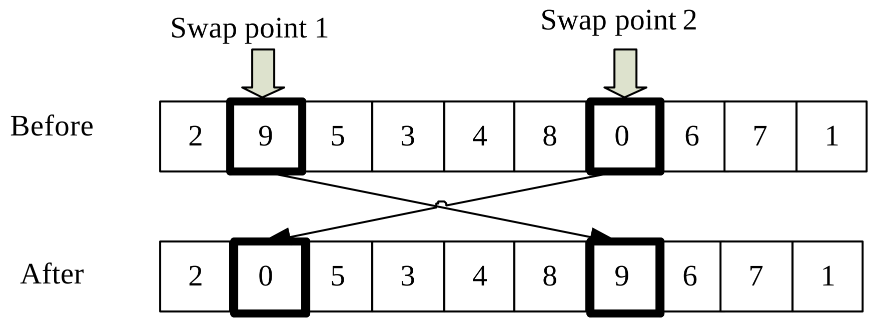


Figure 2.6 the swap mutation

As for the shift mutation, it needs to randomly generate two cut points. We may call it cut point 1 and cut point 2. We move the cut point 2 ahead the position of the range so that it replaces the original cut point 1. Then, shifting all point forward for one space until at the end of element on cut point 2. (Because it has been moved to the place of cut point 1) Thus, the shift mutation is done. The figure 2.7 shows how the shift mutation works which supposes there are 10 departments.

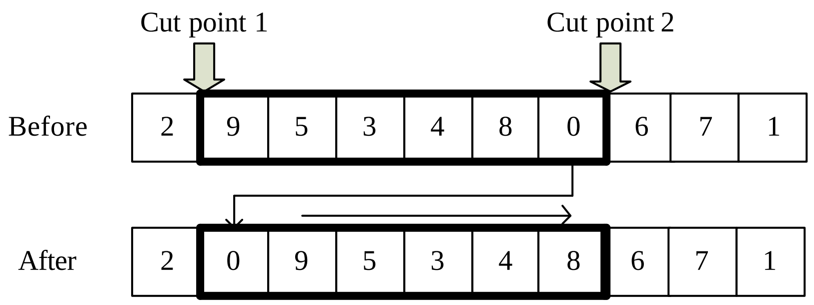


Figure 2.7 Shift mutation

2.7 Replacement and Stopping Criterion

The replacement strategy is total replacement for the QAP problem. For example, if the original population size is m and the offspring is l, then the offspring l will replace m of original population. Finally, the stopping criterion of the algorithm is depended on the number of generations.

2.8 General Procedure of Genetic Algorithm

Because the GA platform developed by the work is not only able to implement simple GA but also extended to several purposes. First, the elitism strategy is employed in the selection stage, we copy a elite individuals into the mating pool. Besides, the concept of multiple crossover and multiple mutation operator are used here. Therefore, the pseudo code are presented as follows.

```
1.  Initialize()
2.  Fitness()
3.  counter ← 0
4.  while counter < Iteration1 do
5.      for i = 1 to ns do
6.          FindPareto()
7.          Fitness()
8.          Elitism()
9.          Selection()
10.         Crossover() //the secondary crossover operator may be implemented
```

```
11.         Mutation() //the secondary mutation operator may be implemented
12.         Replacement()
13.     end for
14.     counter ← counter + 1
15. end while
```

3. Solving the Continuous Problem

The process of GA includes initialization, fitness function, selection, crossover, mutation, population maintenance, and stop rule. By the way, we try two different kinds of GA to do the experiment. The first one is called simple GA. The second is named the modified GA, whose structure is the same in section 2. Moreover, the encode type of the modified GA is encoded in real code. Therefore, there are some different places when we do the crossover and mutation process between the simple GA and modified GA. We will introduce these procedures as follows.

3.1 Initialization

We encode this problem as bit string. To Start with, we generate the initial population randomly. Because this is a continuous problem, we should do precision of bit encoding: $\frac{a-b}{2^l-1}$

Where

a : upper bound
b : lower bound
l : chromosome length

Then, we decode by $b+\frac{a-b}{2^l-1}\times\sum_{j=0}^{l-1} s_{l-j}\times 2^j$

where $x=(s_1,\dots s_l)$

3.2 Fitness Function

The fitness function in GA is a measure of goodness of a chromosome (solution) to the objective function. The fitness function of an individual chromosome is directly equal to its objective function value as follows:

$$\text{Min}\quad z=(x_1^2+x_2-11)^2+\left(x_1+x_2^2-7\right)^2+0.1\times\left[(x_1-3)^2+(x_2-2)^2\right]$$

x_1: The first dimension

x_2: The second dimension

3.3 Selection

During selection phase, parent solutions are selected from the current population. The selecting method we use is binary tournament method. It is a simple way because it just selects two different parents randomly and compares their fitness value. So the better chromosome is selected into the mating pool.

3.4 Crossover

In this phase, there exists some differences. In simple GA, we chose the single point crossover method. The method works as follows:

Step 1: Select two parents P_1 and P_2 randomly from population.
Step 2: Generate one crossover point cp_1 randomly. Then, exchange two substrings, which defined by two positions, between parents P_1 (1, 1, 1) and P_2 (1, 0, 0) to product C_1 and C_2.

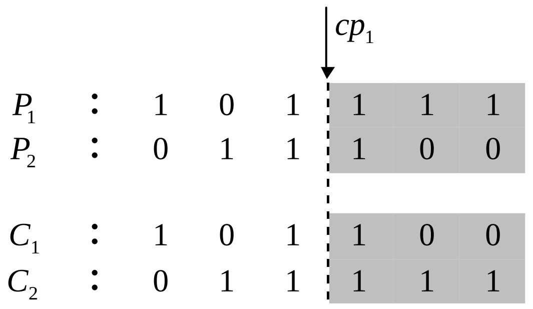


Figure3.1 example of single point crossover

However, in the modified GA, we use arithmetic crossover method and intermediate crossover method because its encoding is in real code. The arithmetic crossover method works as follows:

Step 1: Select two parents P_1 and P_2 randomly from population.
Step 2: Set a a value. The offspring X' will be $a \times X_1 + (1-a) \times X_2$. Shown as the below figure.

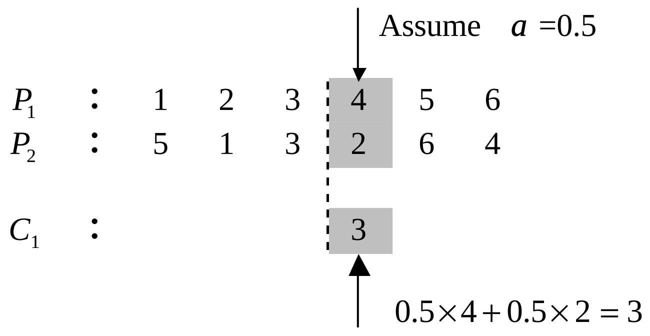


Figure3.2 An example of arithmetic crossover

The intermediate crossover method works as follows:

Step 1: Select two parents P_1 and P_2 randomly from population.
Step 2: The offspring X' will be $X_1 + u \times (X_2 - X_1)$. $u \in U$ [0, 1] Shown as the below figure.

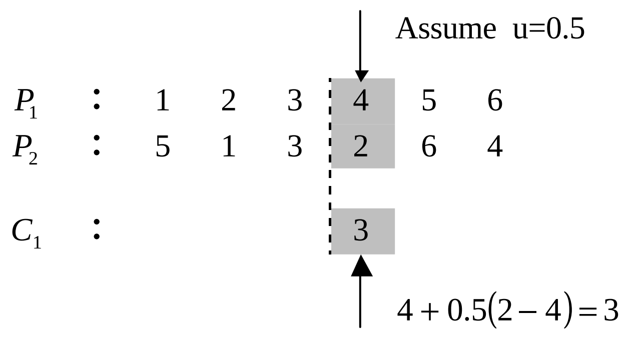


Figure3.3 example of intermediate crossover

3.5 Mutation

In this phase, there still exists some difference. In simple GA, we used bit flip method. The method works as follows. Generate a random number between 1 and N (gene numbers). For example, if we produce the random number i, change the ith allele. Also we can say the original value 1 will change to 0.The original value 0 will change to 1.

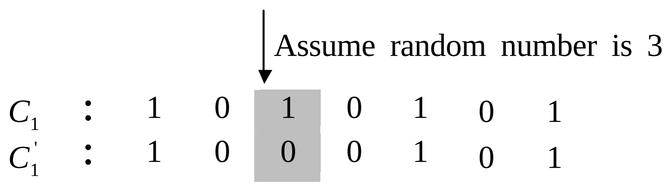


Figure3.4 Example of bit flip

However, in the modified GA, the mutation strategy of real coding for continuous problem uses moving current solution up or down that is depending on random probability. If U(0,1) is larger than 0.5, the current value moves up; otherwise, moves down. The moving length is depended on the range from current position to boundary. The equation of the mutation strategy below:

$$\begin{cases} x_i = x_i + (upBound_i - x_i) \times U(0,1) & if \quad U(0,1) > 0.5 \\ x_i = x_i + (x_i - lwBounds_i) \times U(0,1) & otherwise \end{cases}$$

Hence, if the original x_i is –2.4 and there is a random value U(0,1) = 0.6 for judging the value to go up or go down, the value moves up. Besides, the other random value U(0,1) = 0.1, the x_i -2.4 + (6 – (-2.4))*0.1 = -1.56.

3.6 Population Maintenance

During population maintenance phase, we use two evolutionary strategies to do the experiment. One is comma strategy. It generates 300 offspring from 300 parents and replaces all parents. The other one is plus strategy. It generates 300 offspring from 300 parents and selects the 300 best individuals from the 600 individuals.

3.7 Stop Rule

When maximum number of generations is reached, this experiment will stop.

4. An Object-Oriented Component Design

4.1 Introduction of the Component and Its Interfaces

The second goal of the study is to provide a well-designed callable component to implement GA, which is named OpenGA. OpenGA defines some general interfaces for each procedure of GA and is able to solve different kinds of problems, including the combinatorial problem and continuous problem. Moreover, it not only supports the multiple-objective problem but also the single objective problem. Based on the development platform, the SPGA is implemented by OpenGA. The following table shows these procedures' corresponding interfaces.

Table 4.1 the purposes and its corresponding interfaces

Purposes	Interface
Control the main procedures of GA	MainI
Select better individual into the mating pool	SelectI
Crossover	CrossoverI
Mutation	MutationI
Evaluation of objective functions	ObjectiveFunctionI
To calculate the fitness of each chromosome	FitnessI
A solution class to store chromosome and forms a population	populationI

Take the MainI for example, it is an interface which defines the behavior of main procedures, such as starting GA, initializing a population, selection, mating, mutation, calculating objective values, assigning fitness, and so on. Furthermore, the OpenGA provides interfaces to accept auxiliary crossover and mutation operators. Both of them can be integrated into the standard procedure of crossover and mutation. The multiple crossover and mutation operator may benefit the solution diversity. Then, if we want to add others method, we can simply add it to the additional method. The figure 4 shows the structure of MainI which is presented by UML diagram. It also describes the there are two classes, SingleThreadGA, implement the Main. Moreover, there are some applications will call the interface, such as flow shop, Himmelblau, parallelMachine, QAP_NVR and singleMachine problem. The Himmelblau is a single objective continuous function. Therefore, the OpenGA is applicable to solve different kinds of problems.

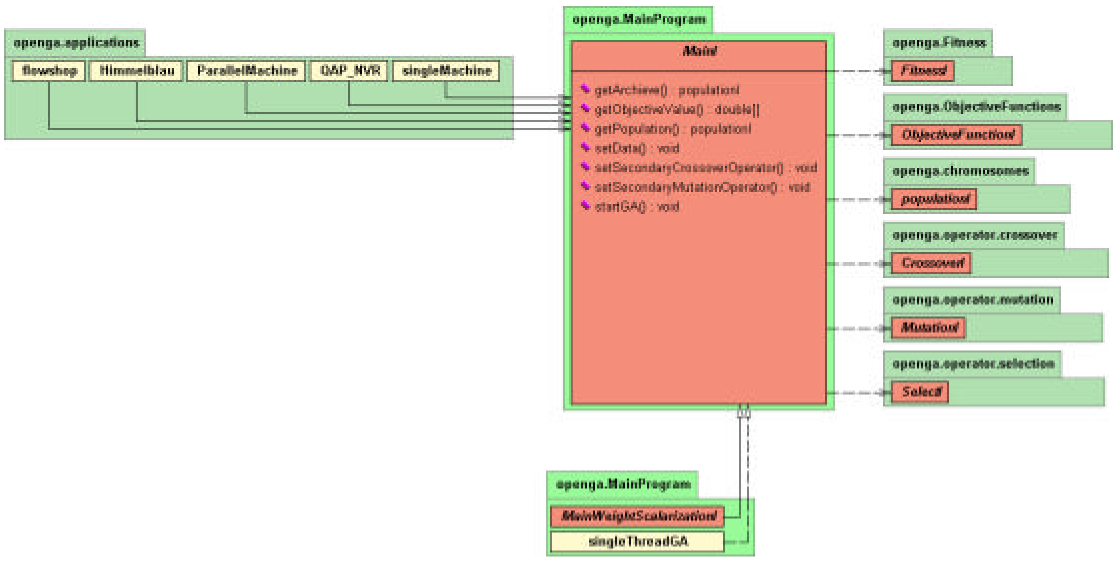


Figure 4.1 the main program of OpenGA

4.2 Classes of the Component

Although the interfaces define the expected behavior of the object, it can't do anything because there is no code in the interface. However, the class programs will implement these interfaces and execute specific actions. The table 2 points out the purposes of different classes and the interfaces that they implement.

Table 4.2 the classes in OpenSA

Purpose	Class name	Implements the interface
The main procedures of GA when solving the combinatorial problem	SingleThreadGA SingleThreadGAwithMultiple-CrossoverAndMutation FixWeightScalarization SPGAwithSharedParetoSet	MainI
Select better individual into the mating pool	binaryTournament rouletteWheel trinaryTournament varySizeTournament	SelectI
Crossover	twoPointCrossover2 PMX ArithmeticCrossover intermediateCrossover	CrossoverI

Mutation	InverseMutation ShiftMutation swapMutation	MutationI
	realValueMutation	RealMoveI
Evaluation of objective functions	ObjectiveFunctionContinuous	ObjectiveFunctionI
	ObjectiveMakeSpan ObjectiveTardiness ObjectiveTardinessForFlowShop	ObjectiveFunctionScheduleI
To calculate the fitness of each chromosome	GoldbergFitnessAssignment FitnessByScalarizedM_objectives singleObjectiveFitness	FitnessI
A solution class to store chromosome and forms a population	population	populationI

Take the singleThreadGA for instance, the class implements the solution MainI, so the methods (setData, setSecondaryCrossoverOperator, setSecondaryMutationOperator, startGA,…) defined by MainI are all implemented at singleThreadGA. The figure 5 shows the UML diagram of singleThreadGA.

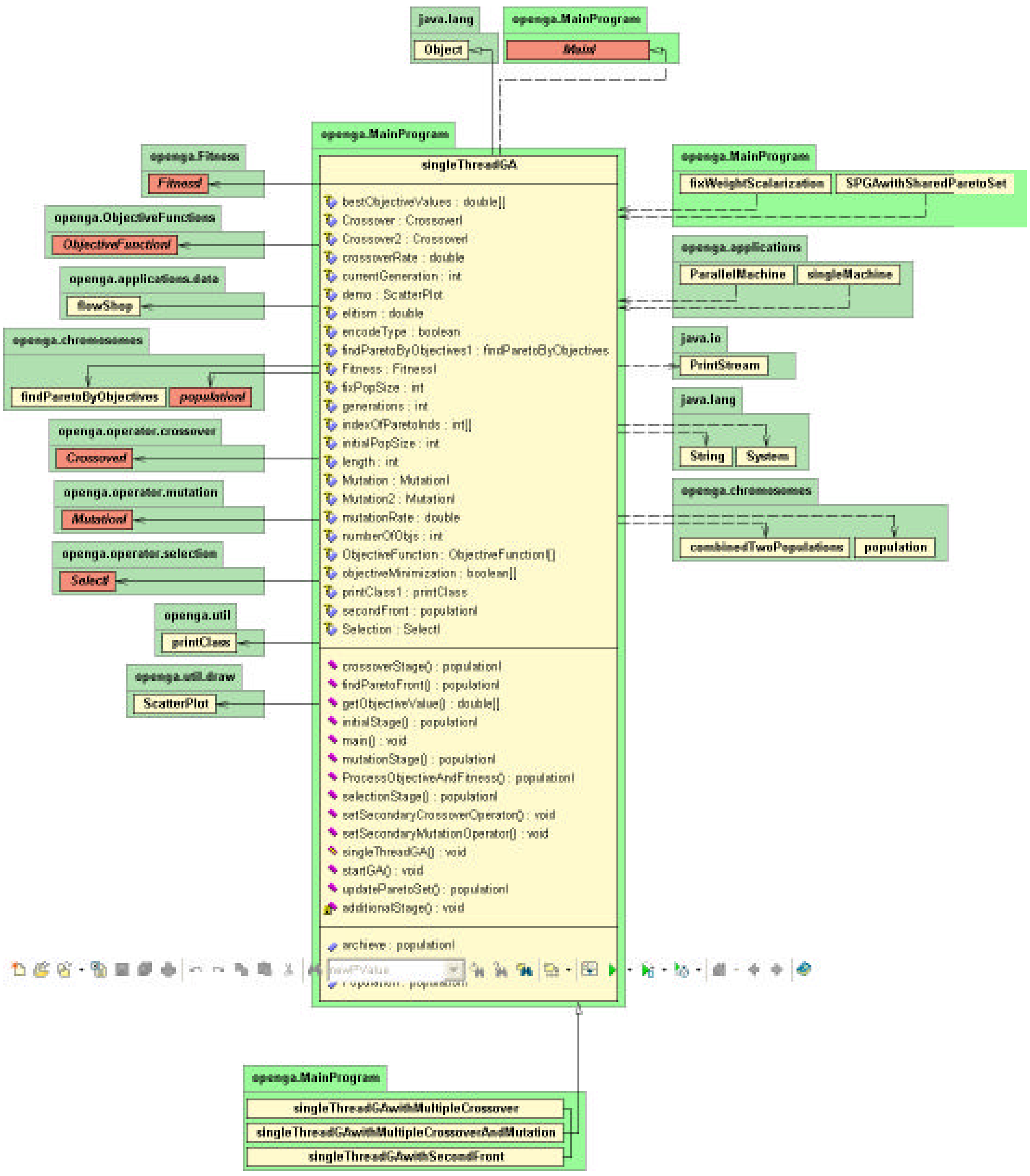


Figure 4.2 The UML diagram of Main procedure of GA

5. Experiment Result

The experimental result includes the QAP problem of Nug30 and the continuous problem of Himmelblau function. The changing of crossover rate, changing mutation rate, and changing the random number seed are considered in both cases. Moreover, the work uses some statistical skills to analyze the experiment result.

The working environment is as follows:

OS	Microsoft Windows XP Professional Service Pack 1
CPU	Intel Pentium 4, 2.8 GHz
RAM	1024 MB (DDR SDRAM)
Coded by	MATLAB 7.0 & JAVA (We compile the Java program into the native binary code. It’s done by a Java JITs compiler.)

From above description, there are three main factors. To simplicity, they are named as follows:

X: The effect from changing crossover rate whose treatments are 3
Y: The effect from changing mutation rate whose treatments are 3
Z: The effect from changing random seed whose treatments are 10

Hence, the Statistics model can be represented as following:
$T = X + Y + X\,Y + Z + X\,Z + Y\,Z$

Where

T: The objective value for QAP or Himmelblau function

From the Statistics model shows we consider the interaction between two factors. However, since the probability of interaction among the three factors is rare, it is not taken into consideration.

5.1 QAP Experimental Result

Before the experiment begins, we have some parameters settings as follows:

- Generation = 1000
- Initial population size = 500
- Population size = 300

- Crossover rate = {1, 0.8, 0.6}
- Mutation rate = {0.05, 0.2, 0.4}
- Random number seed = 500~509

The experiment result of QAP shows at the table 5.1 which has 90 combinations.

Table 5.1 the experimental result of QAP experiment

Counter	Seed	Crossover Rate	Mutation Rate	Obj Value	Time
0	500	1	0.05	3188	18.86
1	500	1	0.2	3138	18.172
2	500	1	0.4	3198	18.547
3	500	0.8	0.05	3140	17.469
4	500	0.8	0.2	3134	17.656
5	500	0.8	0.4	3163	17.703
6	500	0.6	0.05	3138	16.938
7	500	0.6	0.2	3141	16.985
8	500	0.6	0.4	3143	17.625
9	501	1	0.05	3149	18.187
10	501	1	0.2	3291	17.938
11	501	1	0.4	3192	18.359
12	501	0.8	0.05	3197	17.437
13	501	0.8	0.2	3120	17.359
14	501	0.8	0.4	3192	17.719
15	501	0.6	0.05	3190	16.89
16	501	0.6	0.2	3257	17.032
17	501	0.6	0.4	3191	17.234
18	502	1	0.05	3227	17.922
19	502	1	0.2	3115	18.109
20	502	1	0.4	3224	18.766
21	502	0.8	0.05	3188	17.531
22	502	0.8	0.2	3210	17.484
23	502	0.8	0.4	3163	17.937
24	502	0.6	0.05	3142	16.985
25	502	0.6	0.2	3128	16.656
26	502	0.6	0.4	3202	17.172
27	503	1	0.05	3157	17.422
28	503	1	0.2	3196	17.813
29	503	1	0.4	3246	17.828

30	503	0.8	0.05	3273	17.047

Counter	Seed	Crossover Rate	Mutation Rate	Obj Value	Time
31	503	0.8	0.2	3129	17.343
32	503	0.8	0.4	3198	17.454
33	503	0.6	0.05	3174	16.531
34	503	0.6	0.2	3194	16.735
35	503	0.6	0.4	3165	16.89
36	504	1	0.05	3250	17.579
37	504	1	0.2	3104	17.781
38	504	1	0.4	3212	18.172
39	504	0.8	0.05	3170	17.719
40	504	0.8	0.2	3215	17.703
41	504	0.8	0.4	3141	17.938
42	504	0.6	0.05	3166	17.047
43	504	0.6	0.2	3127	17.093
44	504	0.6	0.4	3221	17.266
45	505	1	0.05	3158	18.25
46	505	1	0.2	3123	18.296
47	505	1	0.4	3189	18.703
48	505	0.8	0.05	3167	17.563
49	505	0.8	0.2	3190	17.765
50	505	0.8	0.4	3159	17.782
51	505	0.6	0.05	3137	17.297
52	505	0.6	0.2	3143	17.078
53	505	0.6	0.4	3148	17.641
54	506	1	0.05	3203	18.562
55	506	1	0.2	3152	18.328
56	506	1	0.4	3221	18.703
57	506	0.8	0.05	3168	17.672
58	506	0.8	0.2	3116	17.922
59	506	0.8	0.4	3135	18
60	506	0.6	0.05	3162	17.203
61	506	0.6	0.2	3195	17.11
62	506	0.6	0.4	3175	17.39
63	507	1	0.05	3213	18.219
64	507	1	0.2	3263	18.406
65	507	1	0.4	3205	18.782
66	507	0.8	0.05	3199	17.687

Counter	Seed	Crossover Rate	Mutation Rate	Obj Value	Time
67	507	0.8	0.2	3188	17.984
68	507	0.8	0.4	3192	18.188
69	507	0.6	0.05	3141	17.093
70	507	0.6	0.2	3162	17.219
71	507	0.6	0.4	3177	17.718
72	508	1	0.05	3226	18.234
73	508	1	0.2	3142	18.5
74	508	1	0.4	3208	18.563
75	508	0.8	0.05	3147	17.75
76	508	0.8	0.2	3155	17.656
77	508	0.8	0.4	3143	18.156
78	508	0.6	0.05	3254	17.547
79	508	0.6	0.2	3117	17.391
80	508	0.6	0.4	3177	17.484
81	509	1	0.05	3222	18.031
82	509	1	0.2	3124	18.281
83	509	1	0.4	3166	18.609
84	509	0.8	0.05	3147	17.688
85	509	0.8	0.2	3199	18
86	509	0.8	0.4	3227	18.203
87	509	0.6	0.05	3147	17.109
88	509	0.6	0.2	3144	17.422
89	509	0.6	0.4	3144	17.5

First, we observe every individual value by the scatter plot.

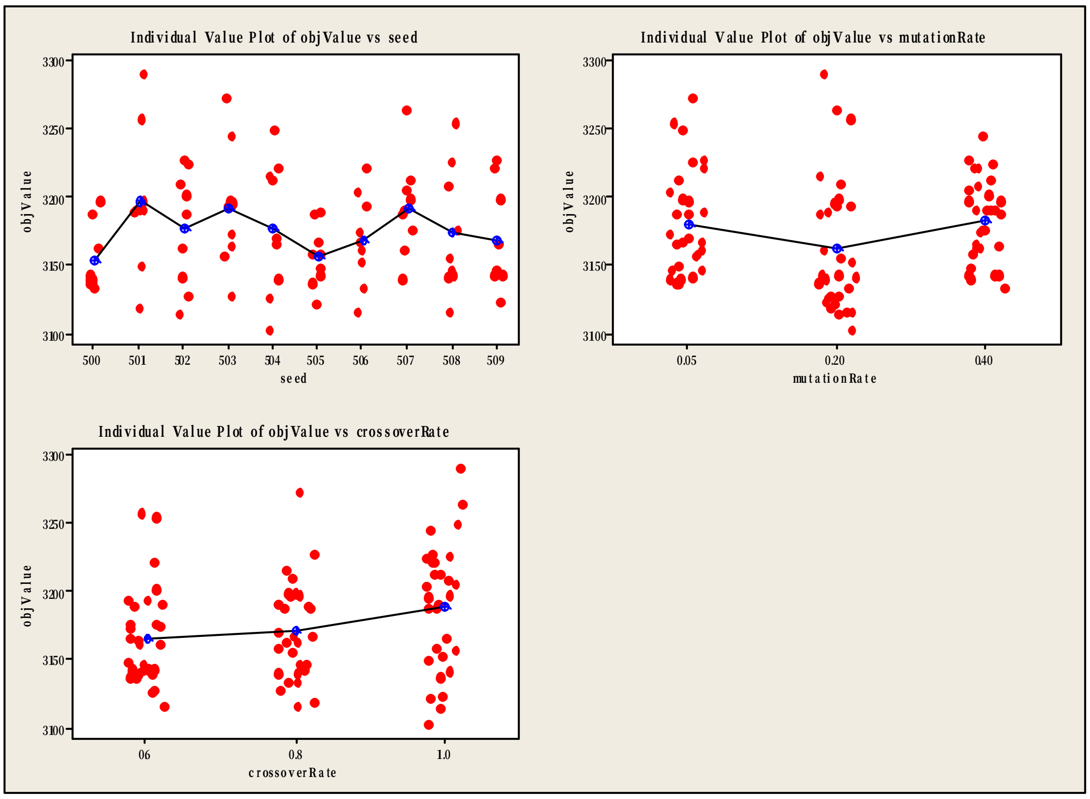


Figure 5.1 scatter plots of QAP

From above figure, we can get some information:

- In random seed scatter plot, we cannot find any rule from it. In other words, it seems different seeds don’t influence obj value.
- In mutation rate scatter plot, we can find when mutation rate = 0.2, dots spread more widely. On the other hand, dots are highly concentrated when mutation rate =0.4. In view of average obj value, it seems no difference among the three mutation levels we set.
- In crossover rate scatter plot, we can find when crossover rate = 1, dots spread more widely. In view of average obj value, it still seems no difference among the three crossover levels we set.

```
Analysis of Variance for objValue

Source                       DF      SS    MS     F      P
seed                          9   17973  1997  1.05  0.424
crossoverRate                 2    8944  4472  2.34  0.110
mutationRate                  2    7230  3615  1.89  0.165
seed*crossoverRate           18   18713  1040  0.54  0.915
seed*mutationRate            18   16740   930  0.49  0.947
crossoverRate*mutationRate    4    4519  1130  0.59  0.670
Error                        36   68684  1908
Total                        89  142804
```

From above ANOVA result, we find that all p-value are larger than 0.05. In another word, we can’t find any factor that can cause significant difference.

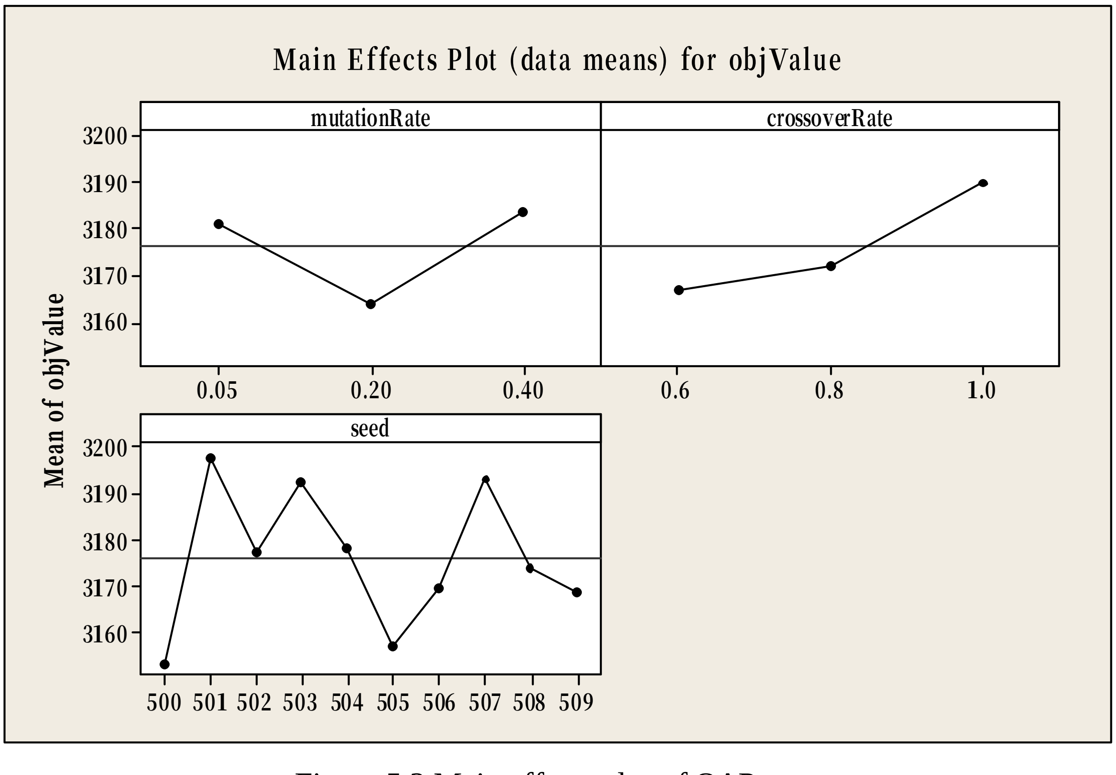


Figure 5.2 Main effects plot of QAP

Figure 5.2 shows when we apply the mutation rate = 0.2 and crossover rate = 100, it may be able to find better solution than others treatments.

According to above analysis, the random seed is deleted in order to estimate the variance from other two factors more precisely. The revised statistics result is as follows.

```
Analysis of Variance for objValue

Source                        DF      SS    MS     F      P
crossoverRate                  2    8944  4472  2.97  0.057
mutationRate                   2    7230  3615  2.40  0.097
crossoverRate*mutationRate     4    4519  1130  0.75  0.561
Error                         81  122110  1508
Total                         89  142804
```

From above ANOVA result, we also find that all p-value are larger than 0.05. In another word, the experiment model still not causes any significant difference.

Here, we conduct the other experiment in order to know the effect of elitism and number of generations. The parameters setting is as follows:

- Generation = {400, 700, 1000}
- Population size = 300
- Elitism= {0.05, 0.1, 0.2, 0.3}
- Crossover rate = 0.6
- Mutation rate = 0.2
- Random number seed = 0~9

```
Analysis of Variance for objValue

Source                  DF      SS    MS     F      P
elitism                  3    5933  1978  1.37  0.261
generations              2     704   352  0.24  0.784
seed                     9    5019   558  0.39  0.936
elitism*seed            27   43847  1624  1.13  0.345
generations*seed        18   31255  1736  1.21  0.290
elitism*generations      6    5772   962  0.67  0.676
Error                   54   77753  1440
Total                  119  170283
```

From above ANOVA result, we find that all p-value are larger than 0.05. In another word, we can't find any factor that can cause significant difference.

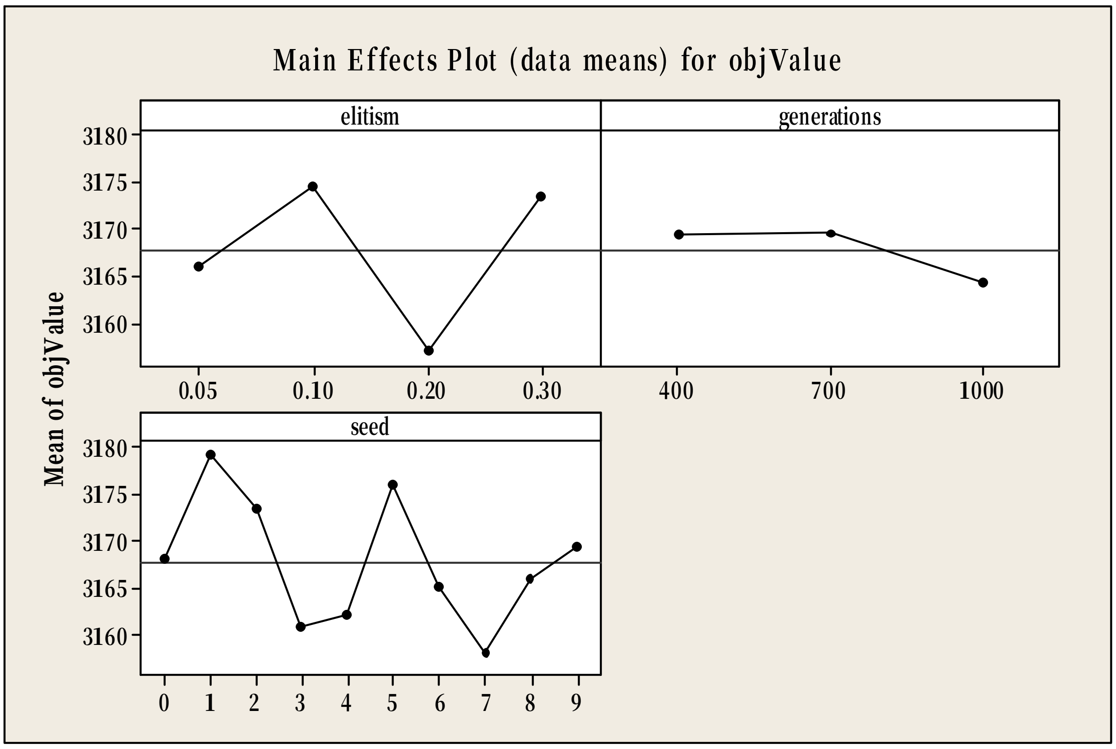


Figure 5.3 New experiment main effects plot of QAP

Figure 5.3 shows when we apply the elitism = 0.2 and generations = 1000, it may be able to find better solution than others treatments.

From above analysis, we still try to ignore the influence of random seed. The result is as follows.

Analysis of Variance for objValue

Source	DF	SS	MS	F	P
elitism	3	5933	1978	1.35	0.261
generations	2	704	352	0.24	0.786
elitism*generations	6	5772	962	0.66	0.684
Error	108	157874	1462		
Total	119	170283			

From above ANOVA result, we also find that all p-value are larger than 0.05. In another word, we can't find any factor that can cause significant difference.

From here, we turn to focus on the relationship between time and the three factors.

```
Analysis of Variance for time

Source               DF        SS        MS         F      P
elitism               3   368.460   122.820   1979.08  0.000
generations           2  2739.830  1369.915  22074.36  0.000
elitism*generations   6    44.116     7.353    118.48  0.000
seed                  9    18.015     2.002     32.26  0.000
elitism*seed         27     5.243     0.194      3.13  0.000
generations*seed     18     1.919     0.107      1.72  0.065
Error                54     3.351     0.062
Total               119  3180.934
```

The ANOVA tells us only the p-value of generation*seed greater than 0.05.In other words, all factors except generation*seed cause significant difference.

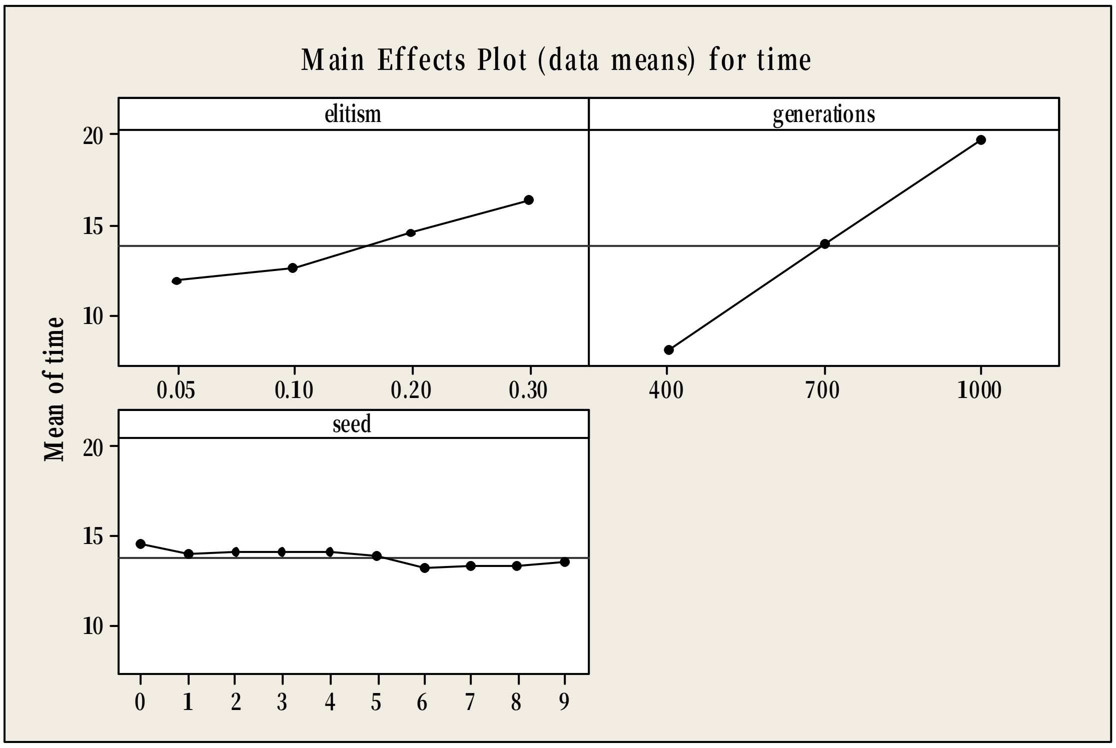


Figure 5.4 New experiments for time main effects plot of QAP

Figure 5.4 shows when we apply the elitism = 0.05 and generations = 400, it may be able to find solution effectively than others treatments.

5.2 Continuous Problem Result

Before the experiment begins, we have some parameters settings as follows:

- Generation = 500
- Initial population size = 500
- Population size = 300
- Crossover rate = {1, 0.8, 0.6}
- Mutation rate = {0.05, 0.2, 0.4}
- Random number seed = 500~519

After the experiment, we find both simple GA and the modified GA have obj value =0. Thus, we just list the modified GA experiment result of continuous problem shows at the table 5.2 which has 90 combinations. You can see the simple GA experiment result of continuous problem in appendix I.

Table 5.2 the experimental result of continuous problem experiment

counter	seed	Crossover	Mutation	X1	X2	Obj Value	Time(sec)
0	500	1	0.05	2.999999097	2.000002356	0.000000	3.016
1	500	1	0.2	2.999998792	2.000004245	0.000000	2.89
2	500	1	0.4	2.999999748	1.999998908	0.000000	2.875
3	500	0.8	0.05	2.999999972	1.999999408	0.000000	2.891
4	500	0.8	0.2	2.999999978	2.00000005	0.000000	2.938
5	500	0.8	0.4	3.000000013	1.999999973	0.000000	2.875
6	500	0.6	0.05	3	2	0.000000	2.75
7	500	0.6	0.2	3	2	0.000000	2.968
8	500	0.6	0.4	3	2	0.000000	2.875
9	501	1	0.05	2.999999059	1.999998786	0.000000	2.891
10	501	1	0.2	2.999998344	2.00001578	0.000000	3.016
11	501	1	0.4	2.999997849	2.000000846	0.000000	2.968
12	501	0.8	0.05	3.000000067	1.999999941	0.000000	2.953
13	501	0.8	0.2	2.99999998	1.999999882	0.000000	2.907
14	501	0.8	0.4	3.000000119	1.999999731	0.000000	2.922
15	501	0.6	0.05	3	2	0.000000	3.39
16	501	0.6	0.2	3	2	0.000000	3.188
17	501	0.6	0.4	3	2	0.000000	3.125
18	502	1	0.05	2.999999929	1.999996283	0.000000	2.937
19	502	1	0.2	3.000000211	2.000000633	0.000000	2.985

20	502	1	0.4	2.999995482	2.000004467	0.000000	2.953
21	502	0.8	0.05	2.999999955	1.999999896	0.000000	2.937

counter	seed	Crossover	Mutation	X1	X2	Obj Value	Time(sec)
22	502	0.8	0.2	3.000000009	1.999999999	0.000000	2.906
23	502	0.8	0.4	3.000000059	1.999999933	0.000000	2.985
24	502	0.6	0.05	3	2	0.000000	2.922
25	502	0.6	0.2	3	2	0.000000	2.859
26	502	0.6	0.4	3	2	0.000000	2.859
27	503	1	0.05	2.999999066	1.999997377	0.000000	2.985
28	503	1	0.2	3.000001852	1.999996783	0.000000	3.031
29	503	1	0.4	2.999998408	2.000000158	0.000000	3.062
30	503	0.8	0.05	2.999999997	2.00000001	0.000000	2.922
31	503	0.8	0.2	2.999999998	2.000000018	0.000000	3.016
32	503	0.8	0.4	2.999999996	2.000000003	0.000000	3.218
33	503	0.6	0.05	3	2	0.000000	3.063
34	503	0.6	0.2	3	2	0.000000	3.062
35	503	0.6	0.4	3	2	0.000000	3.172
36	504	1	0.05	3.000002028	1.999996789	0.000000	3.235
37	504	1	0.2	3.000000076	2.000001914	0.000000	3.109
38	504	1	0.4	2.999999826	2.000000886	0.000000	3.297
39	504	0.8	0.05	2.999999897	1.999999834	0.000000	3.14
40	504	0.8	0.2	3.000000019	1.99999997	0.000000	3.235
41	504	0.8	0.4	3.000000006	2.000000011	0.000000	3.015
42	504	0.6	0.05	3	2	0.000000	3.063
43	504	0.6	0.2	3	2	0.000000	3.078
44	504	0.6	0.4	3	2	0.000000	3.25
45	505	1	0.05	2.999999418	2.000000835	0.000000	3.14
46	505	1	0.2	2.999999397	1.999997856	0.000000	3.109
47	505	1	0.4	3.000002124	1.99999787	0.000000	3.297
48	505	0.8	0.05	2.999999818	2.000000067	0.000000	3.078
49	505	0.8	0.2	2.999999953	2.00000006	0.000000	3.125
50	505	0.8	0.4	2.999999989	1.999999927	0.000000	3.204
51	505	0.6	0.05	3	2	0.000000	3.062
52	505	0.6	0.2	3	2	0.000000	3.14
53	505	0.6	0.4	3	2	0.000000	3.016
54	506	1	0.05	3.000000553	2.000000757	0.000000	3.344
55	506	1	0.2	3.000003807	1.999999333	0.000000	3.125
56	506	1	0.4	2.999998152	2.000003556	0.000000	3.203

57	506	0.8	0.05	2.999999989	1.999999677	0.000000	3.094
58	506	0.8	0.2	2.999999984	2.000000016	0.000000	3.203

counter	seed	Crossover	Mutation	X1	X2	Obj Value	Time(sec)
59	506	0.8	0.4	3.000000036	1.999999892	0.000000	3.11
60	506	0.6	0.05	3	2	0.000000	3.109
61	506	0.6	0.2	2.999999999	2	0.000000	3.109
62	506	0.6	0.4	3	2	0.000000	3.359
63	507	1	0.05	2.999998408	1.999993363	0.000000	3.172
64	507	1	0.2	3.000002072	1.999995053	0.000000	3.422
65	507	1	0.4	3.000001251	1.99999925	0.000000	3.453
66	507	0.8	0.05	3.00000001	2.000000016	0.000000	3.281
67	507	0.8	0.2	2.999999997	1.999999985	0.000000	3.125
68	507	0.8	0.4	2.999999983	2.000000037	0.000000	3.141
69	507	0.6	0.05	3	2	0.000000	3.031
70	507	0.6	0.2	3	2	0.000000	3.328
71	507	0.6	0.4	3	2	0.000000	3.781
72	508	1	0.05	2.999999218	1.999999496	0.000000	3.281
73	508	1	0.2	2.999997909	1.999992512	0.000000	3.219
74	508	1	0.4	3.000000113	1.999997368	0.000000	3.11
75	508	0.8	0.05	2.999999934	2.000000014	0.000000	3.156
76	508	0.8	0.2	3.000000021	1.999999866	0.000000	3.109
77	508	0.8	0.4	2.999999895	1.999999955	0.000000	3.188
78	508	0.6	0.05	3	2	0.000000	3.125
79	508	0.6	0.2	3	2	0.000000	3.015
80	508	0.6	0.4	3	2	0.000000	3.094
81	509	1	0.05	3.000000294	2.000001952	0.000000	3.11
82	509	1	0.2	2.999996894	2.000005954	0.000000	3.265
83	509	1	0.4	2.999999771	2.000000561	0.000000	3.156
84	509	0.8	0.05	2.999999997	2.000000052	0.000000	3.094
85	509	0.8	0.2	2.999999959	1.999999757	0.000000	3.203
86	509	0.8	0.4	2.999999995	1.999999989	0.000000	3.094
87	509	0.6	0.05	3	2	0.000000	2.968
88	509	0.6	0.2	3	2	0.000000	2.907
89	509	0.6	0.4	3	2	0.000000	2.875

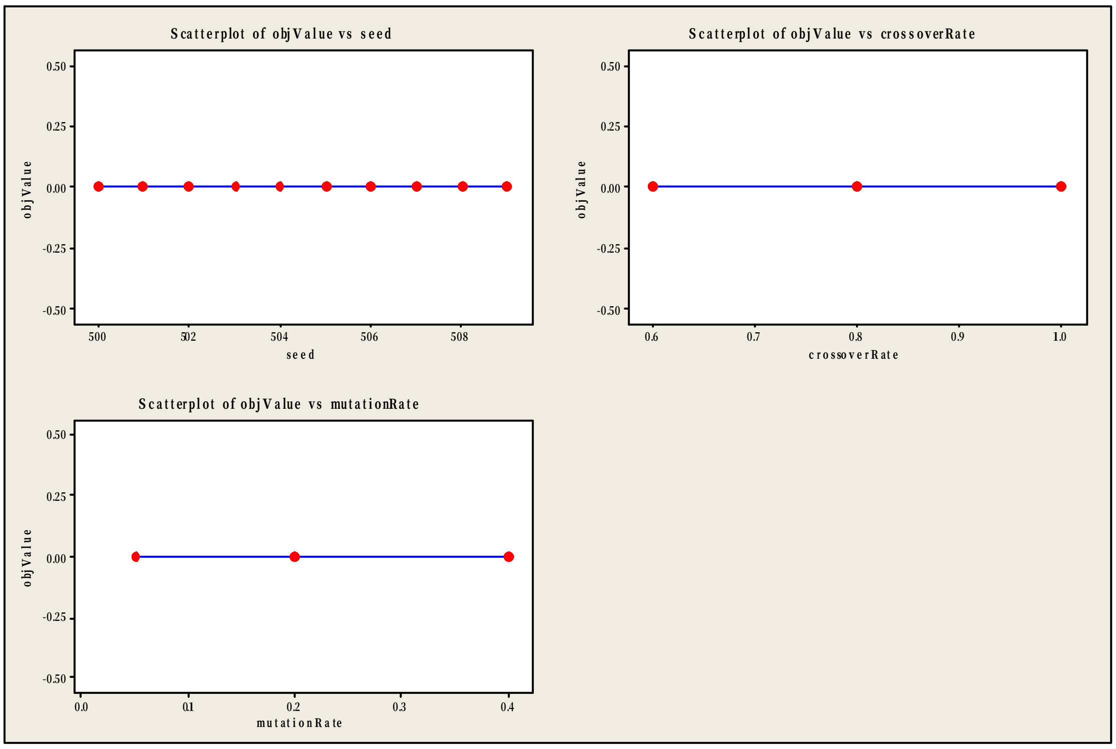


Figure 5.5 scatter plots of continuous problem

We observe every individual value by the scatter plot and obtain some information:

- No matter seed scatter plot, crossover rate scatter plot or mutation rate scatter plot, we can see the obj value all = 0 in each treatment. In other words, we cannot find any factor that can cause significant difference.

6. Discussion and Conclusions

The study does two experiments in combinatorial problem and continuous problem. Besides, in order to obtain better solution quality, there are some procedures, elitism and multiple crossover operators and multiple mutation operators, which are added in the GA process. The elitism is to copy better chromosomes in the external archive and let these individuals to join the mating procedure. The purpose of employing the multiple crossover operators and multiple mutation operators is to do more variation on each chromosome. The effort will increase the diversity of the chromosomes.

From the first experiments of QAP, the study found GA might not be sensitive to parameters, such as crossover rate and mutation rate, which don't cause significant difference. However, after we eliminate the factor of random number seed, the P-value of crossover rate and mutation rate become 0.057 and 0.097 respectively. (In statistics point of view, the factor doesn't cause significant can be ignored.) If we set the a = 0.05, both of them are still not significant but they close to cause the significant difference. Hence, according to the revised ANOVA table, we suggest that to use the crossover rate is 0.6 and the mutation rate is 0.2. It may accordance to some researches' finding about "to apply the higher crossover rate with lower mutation rate or lower crossover rate with higher mutation rate."

The study further examines the effect of elitism and generations in the QAP problem. Besides, the objective value and implementation are considered in the statistic model. As for the number of elite, we can't explain how many elites are required during selection. It may need further experiment to know the truth. However, the time is significant when applying different proportion elite. The larger elite size, it causes the higher computational time. According the finding, we may not select too many elites in the external archive.

Then, although the generation 1000 is slightly better than 400 and 700 generations in average, the longer generation causes more time to calculate the solutions in generations. It is not necessary to run 1000 generation. So the work suggest that we can use 400 generation to solve the problem size which is 30 because 400 generations is as good as 1000 generations and it's enough to obtain satisfactory solution quality.

The study compares the simple GA and the modified genetic algorithm in continuous problem, which shows the simple GA is as good as modified GA. The reason may be the problem that is easy to solve, the more complex approach can't present its effort clearly. Consequently, to test one testing function is not always enough, the later study should test different kind of functions to clarify the effect of modified GA in continuous problem.

Finally, the work designs and develops a component which is called OpenGA. It has been proved that it's able to solve the combinatorial problem and continuous problem and both of them are single objective problem. Then, the ability of the OpenGA can be extended to solve the multiple objectives problem. Because the applications had been developed to solve scheduling problem, they may be shown in the final term project.