Genetic Algorithms for Bridge Maintenance Scheduling

Yan ZHANG

Master Thesis

1st Examiner: Prof. Dr. Hans-Joachim Bungartz

2nd Examiner: Prof. Dr. rer.nat. Ernst Rank

Assistant Advisor: DIPL.-ING. Katharina LUKAS

Computational Science and Engineering (Int. Master's Program)

Technische Universität München

September, 2010

DECLARATION

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Abstract

Maintenance for infrastructural buildings, e.g. bridges, should have minimal impact on the quality of traffic flow. Genetic algorithm is applied for solving infrastructure maintenance schedule problem, while different scenarios are created randomly, and better solutions are found by crossover between two scenarios and random mutation. NSGAII was presented to deal with multi-objective optimization problem. By this method, a Pareto optimal set can be obtained, and thus the decision maker can choose a schedule which is most suitable for the maintenance.

CONTENTS

Chapter 1   Introduction and Overview

1.1 Scope of Work

Maintenance for infrastructural buildings, e.g. bridges, should have minimal impact on the quality of traffic flow. Particularly in urban areas, the question is focused on which bridges can be repaired simultaneously with the least disturbances of the traffic. This paper is aimed at getting the good (minimal traffic delay time) schedules of bridges maintenance by genetic algorithms.
Genetic algorithms are generally considered to be robust techniques of almost universal application. This is due to the fact that they can be implemented on a wide variety of problems with minimal adaptation, but are likely to be much less efficient than the highly tailored problem-specific algorithms [2].
Different scheduling scenarios are created randomly and analyzed according to their quality (impact on the traffic, maintenance budget, compromise with other construction on the bridges, etc.).These qualities are so called objective values. By crossover between two scenarios and random mutations it is tried to find particularly suitable representations of the maintenance scenarios.
In this part the field of genetic algorithms is briefly introduced. The main components of the genetic algorithms are presented. Since single objective genetic algorithm (SOGA) is the base of further genetic algorithms, SOGA is presented in next chapter. The maintenance schedules of infrastructural building (focus on minimizing traffic delay time) are also created by SOGA. Chapter 3 introduces multi-objective genetic algorithm (MOGA) and NSGAII (an improved version of MOGA), and analyses the numerical model with two and three objective functions.

1.2 Introduction to Genetic Algorithms (GA)

Genetic algorithms are a class of probabilistic optimization algorithms [3], which are inspired by the biological evolution process. GA uses concepts of “natural selection” and “genetic inheritance” (Darwin 1859) [4], since in nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones, and was originally developed by John Holland (1975).
GA is based on an analogy with the genetic structure and behavior of chromosomes within a population of individuals using the following foundations [5]:
•Individuals in a population compete for resources and mates.
•Those individuals most successful in each 'competition' will produce more offspring than those individuals that perform poorly.
•Genes from `good' individuals propagate throughout the population so that two good parents will sometimes produce offspring that are better than either parent.
•Thus each successive generation will become more suited to their environment.

GA process is stochastic, i.e., even the very weak individuals have the chance to survive while normally the fitter ones have bigger chance to be selected. The general procedural of GA is shown in Fig.1.1. Initially, a group of individuals (population) is created randomly, and then some individuals are selected as parents, thereafter by recombination and mutation, offspring are created. Applying survivor selections within old population and offspring, new population is generated.

```
BEGIN
    INITIALISE  population;
     REPEAT UNTIL( TERMINATION CONDITION is satisfied) DO
        EVALUATE ;
        PARENT SELECTION;
        CROSSOVER;
        MUTATE;
       SURVIVOR SELECTION;
    OD
END
```

Fig.1.1 General Procedural of GA

1.3 Components of Genetic Algorithms

In this section some of the main components of genetic algorithm are briefly described.

1.3.1 Representation

The mapping from real world model to the genetic individuals is called representation. A phenotype, or individual, is the expression of a specific trait, such as eyes colors, based on gene. Genotype, or chromosome, is a class of organisms having the same genetic constitution. The phenotype can be very different from the genotype.
There are a lot different ways of representation in GA, typically binary representation and integer representation which are shown in detail in next chapter.

1.3.2 Population

The population is a group of individuals (is also called chromosome) which is subject to evolution generation by generation. The diversity of a population is a measure of the number of different solutions present [6]. One principle of GA is density preservation, i.e., not only high-quality individuals participate the evolution, but low-quality ones also have positive chance to attend it.
In multi-objective genetic algorithms (chapter 3), special techniques are used for diversity preservation.

1.3.3 Fitness Function

Fitness function, or objective function, is a quality measurement in GA. If there is only one fitness function, it is called single-objective GA (SOGA). The goal of SOGA is usually associated with searching the maximum (or minimum) fitness function value. In reality, there is often more than one objective to achieve. A problem with more than one objective function can be solved by multi-objective GA (MOGA). Since it is impossible to get the maximum (or minimum) values of different fitness functions at the same time in most cases, the goal of MOGA is changed to find the compromised objective function values.

1.3.4 Selection

A parent is an individual selected for creating offspring. Selections based on fitness proportion are ranking selection, roulette wheel selection and tournament selection. The mechanism of these selections is shown in next chapter (2.1). Selection called after having generating the offspring is named survivor selection. It chooses individuals who will go to the next generation. In SOGA, fitness-based survivor selection has the same mechanism as parent selection, while age-based survivor selection making the selection only in the new individuals. In MOGA, survivor selection is focus on not only offspring but also the current population group, the mechanism is shown in chapter 3 (3.5.3).

1.3.5 Variation

Mutation and crossover are two variation operators which create new individuals from old ones. They are stochastic operators. It is mentioned in [20], that the variation operators could be problem-specific. In the next chapter, we use different variation operators for the binary represented problem and the integer represented problem.

Chapter 2 Single Objective Genetic Algorithms (SOGA)

Part 1 Binary representations

The binary representation is one of the earliest representations in the GA history. In this paper the genotype consists of a bit-string that 1 represents the performing of the maintenance and 0 represents not performing. Maintenance Agencies make the maintenance schedules for the next five year and only five bridges per year should be under constructions. They anticipate all possible maintenance locations and try to minimize traffic delay time.
A group of population is generated when simulation starts, and every individual genotype, is a set of random binary numbers. The length of chromosome keeps constant in the whole group. For example, a chromosome (length L = 20) can be illustrated as:

10001010100011100101

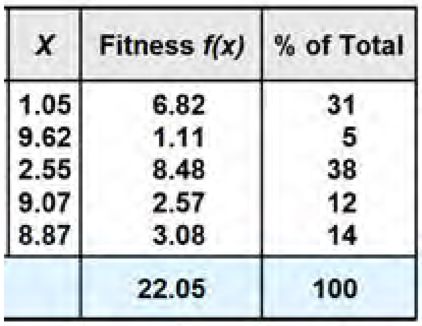
For this binary-represented maintenance problem, we set the length of chromosome as 500, since there are totally 100 bridges in the list and the schedule is planned for the next 5 years. The general scheme of the binary-represented SOGA maintenance problem is the same as in Fig.1.1.

2.1 Parent Selection

For the binary-represented SOGA maintenance problem we use roulette wheel selection, tournament selection or ranking selection

2.1.1 Roulette Wheel Selection

A very simple example is presented here to introduce the basic concepts of roulette wheel selection (the example data is taken from [1]). Fig.2.1 lists a population of 5 individuals with fitness function $f(x) = -\frac{1}{4}x^2 + 2x + 5$.



X	Fitness f(x)	% of Total
1.05	6.82	31
9.62	1.11	5
2.55	8.48	38
9.07	2.57	12
8.87	3.08	14
	22.05	100

Fig.2.1 Example for Roulette Wheel Selection

These individuals, in the first column x, are ranged between 0 and 10. The fitness values are then taken as the function of x. We can see from the list (column Fitness f(x)) that individual No. 3 is the fittest and No. 2 is the weakest. Summing these fitness values we can apportion a percentage total of fitness. This gives the strongest individual a value of 38% (selection probability) and the weakest 5% (as show in Fig. 2.2)

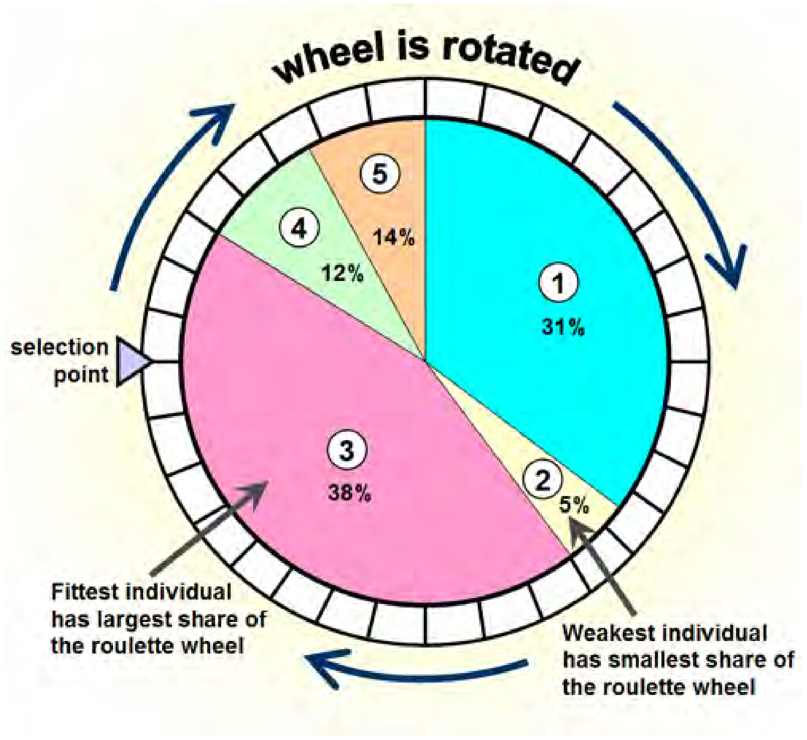

wheel is rotated
5
14%
4
12%
1
31%
selection point
3
38%
2
5%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Fig.2.2. Roulette wheel approach: based on fitness (compare to [1])

We assume the algorithm to select λ members, based on selection probability, from the set of μ individuals into a mating pool. Every time randomly create a number p in the range of [0, 1], then choose an individual whose selection probability is larger than p. In this paper, for binary represented SOGA, λ = 1, μ = N, the outlines of the algorithms are given in Fig.2.3.

```
BEGIN
      Pick a random value p uniformly from [0,1];
      FOR I = 0 TO N  DO
         sum[I] += fitness[I]/total fitness;
         IF ( sum[I] > p ) DO
              Select individual[I];
          BREAK;
          OD
     OD
END
```

Fig.2.3. Pseudo code for the roulette wheel selection

2.1.2 Ranking Selection

The ranking selection sorts the population into ascending order of fitness and then simply assigns a fitness score based on its position in the ladder. So if a genome ends up last it gets score of zero, if best then it gets a score equal to the size of the population. Thereafter individuals are selected according to their fitness scores instead of fitness values, and each individual has the same chance to be selected. The Procedure is shown below in Fig.2.4.

```
BEGIN
      Sort population into ascending order according to fitness;
      Assign new fitness numbers according to the genome's position on this new fitness 'ladder';
      Roulette wheel selection based on new fitness numbers;
END
```

Fig.2.4. Pseudo code for ranking selection

2.1.3 Tournament Selection

The tournament selection is an operator that selects λ fittest members from tournament with size k. First pick k individuals randomly from the mating pool, then select the first λ individuals which have better fitness values. It was shown in [6] that for binary tournaments if k=2, tournament selection works best. The procedure is shown in Fig.2.5.

```
BEGIN
      Set current_member=0;
      WHILE (current_member < μ) DO
            Pick k random individuals from mating_pool;
           Select the fittest one ;
           current_member ++ ;
      OD
END
```

Fig.2.5. Pseudo code for the tournament selection algorithm

2.2 Crossover

We use one-point crossover and n-point crossover recombination operator. They all start from two parents and create two children.

2.2.1 One-Point Crossover

One-point crossover works by choosing a random number in the range [0, L-1] (with L the length of the chromosome), and then splitting both parents at this point and creating the two children by exchanging the tails [6] (Fig. 2.6) (see also [6]).

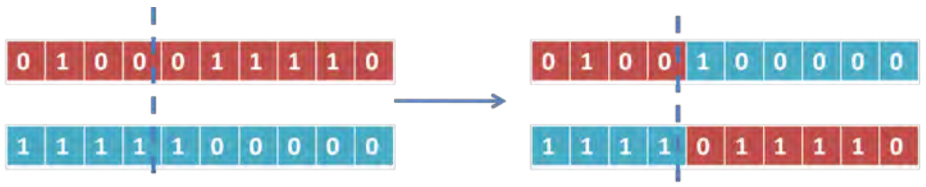


Fig.2.6. One-Point Crossover

2.2.2 N-Point Crossover

One-point crossover can easily be generalized to n-point crossover, where the representation is broken into more than two segments of contiguous genes and then the offspring are created by taking alternative segments from the two parents [4]. The following Fig.2.7 illustrated for n = 2.

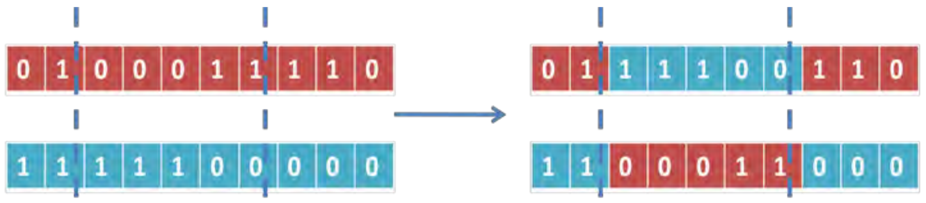


Fig.2.7.N_Point Crossover

2.3 Mutation

2.3.1 Mutation for Binary Representations

We use two kinds of mutation operators:
•flip all bits with mutation rate P_0 (in this paper for binary represented GA, P_0=0.002, since 500 (chromosome length) x 0.002 = 1, which means in average each time one gene is chosen to mutate);
•flip one random bit.

2.3.1.1 Flip all Bits with Mutation Probability P_0

Each bit has a small probability P_0 to flip (i.e., from 0 to 1 or 1 to 0). In Fig.2.8 this is illustrated for the case where the fourth and seventh genes are flipped.

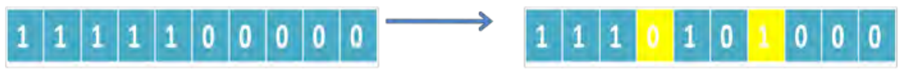


Fig.2.8. Bitwise mutation for binary encodings

2.3.1.2 Flip one Random Bit

We must flip one gene randomly and flip only one gene in each individual as shown in Fig.2.9.

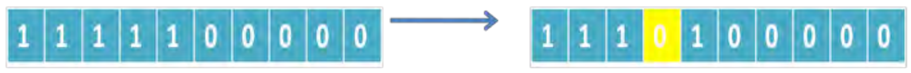


Fig.2.9. Flip one random gene

2.4 Survivor Selection

2.4.1 Age-Based Replacement

In age-based replacement scheme, the fitness of older generation population is not taken into account during the selection of which individuals to go to the next generation, rather they are designed so that each individual exists in the population for the same number of GA iteration [6], i.e., the oldest individuals are eliminated from the population after generation.

2.4.2 Fitness-Based Replacement

Fitness-based replacement strategy is proposed for choosing μ individuals from λ + μ members, which consist of parents and offspring going forward to the next GA iteration [6], the mechanisms used here are principally the same as in parent selection. The selection scheme used in test problem 2.5.1 is tournament selection. Individuals having better fitness scores have bigger chance to enter the next generation.

2.5 Simulation Results

2.5.1 Test Problem

We first describe the test problem. The numerical example is based on the data shown in Fig. 10 and 11. It consists of 100 bridges (index starts from 0 to 99) whose current ages and conditions are listed in Fig.2.10 and 2.11. The conditions of the bridges are depends on their ages. Condition 1 is the best while condition 6 means that the bridge is likely to collapse. None of the bridges should be over condition 6 during the considered time span of 5 years. A maintenance resets the respective bridge to condition index 1.

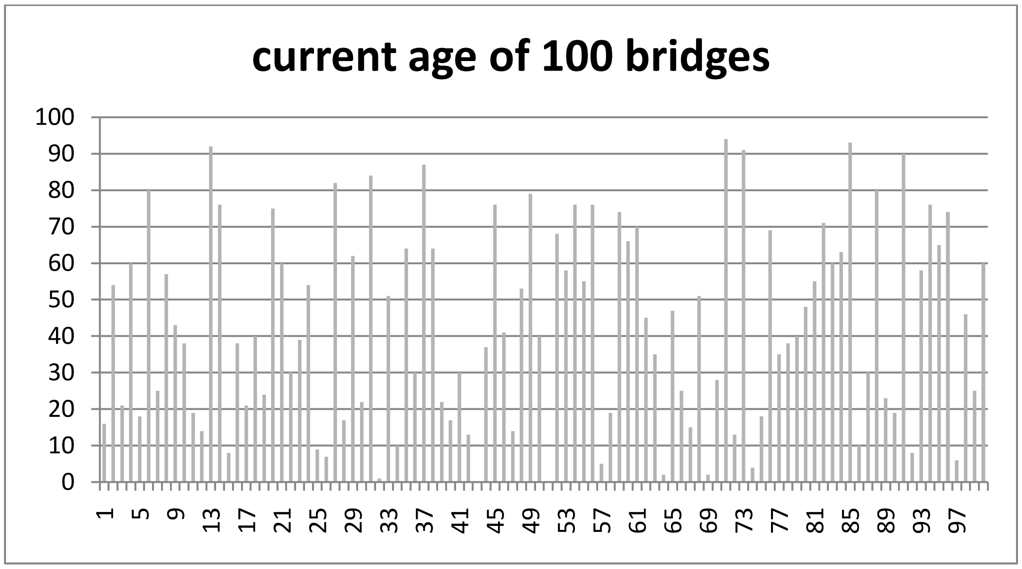


Fig.2.10. Current ages of the bridges

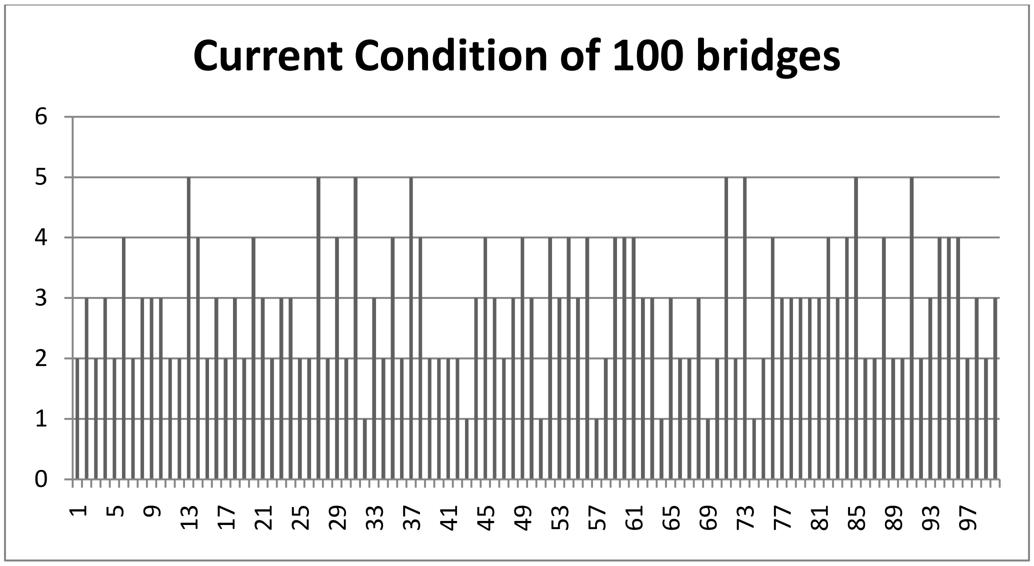


Fig.2.11. Current condition of the bridges

The testing case has one objective function (fitness function) and two constraints. The problem is described in Table 2.1.

Table2.1. Test Problem

Objective Functions (Fitness Functions)	min f(x)=$\max_{i=0,..,4}(\sum_{j=0}^{j=99} gene[i \times 100 + j] \times waitingtime[j])$
Constrains	1. condition[j] < 6 j=0,..,99; 2. Only 5 bridges are under maintenance each year.
Termination Conditions	The best solutions keep unchanged ($\sigma \leq 0.0001$) in 10 successive generation.

All test settings (with different approaches and parameter settings) are run 50 times for a maximum of 1000 generation with a population size 100. We use a crossover probability of P_1 = 0.7 and a mutation probability of P_0 = 0.002.

2.5.2 Performance Measures

Fig.2.12 shows the performance of different parent selection schemes: roulette wheel selection, tournament selection, ranking selection. The variation operators were fixed with one-point crossover and flip one random Bit mutation. From this figure, we can see that generally ranking selection and tournament selection have better performance than roulette wheel selection (this conclusion is only suit for test case in 2.5.1).

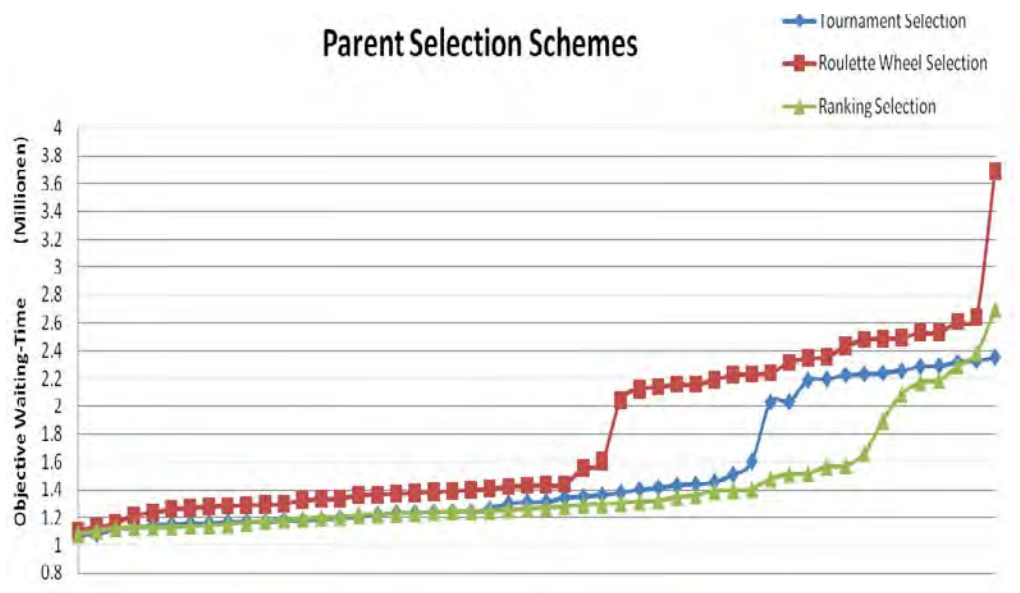


Fig.2.12. Different Parent Selection Schemes

Fig.2.13-Fig.2.15 presents the process of convergence of different parent selection schemes. Fig.2.13 shows that roulette wheel selection converges since generation 5, and keeps stable since the 23rd generation. Fig.2.14 shows that convergence starts at the 3rd generation in tournament selection and becomes stable at generation 28. From Fig.2.15, we can see that ranking selection becomes stable very quick at the 8th generation.

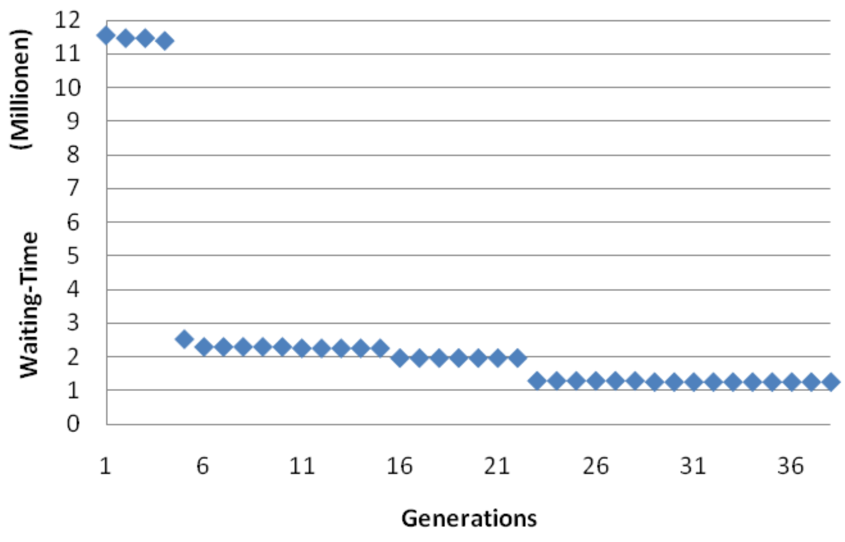


Fig.2.13. Convergence of GA search using roulette wheel selection

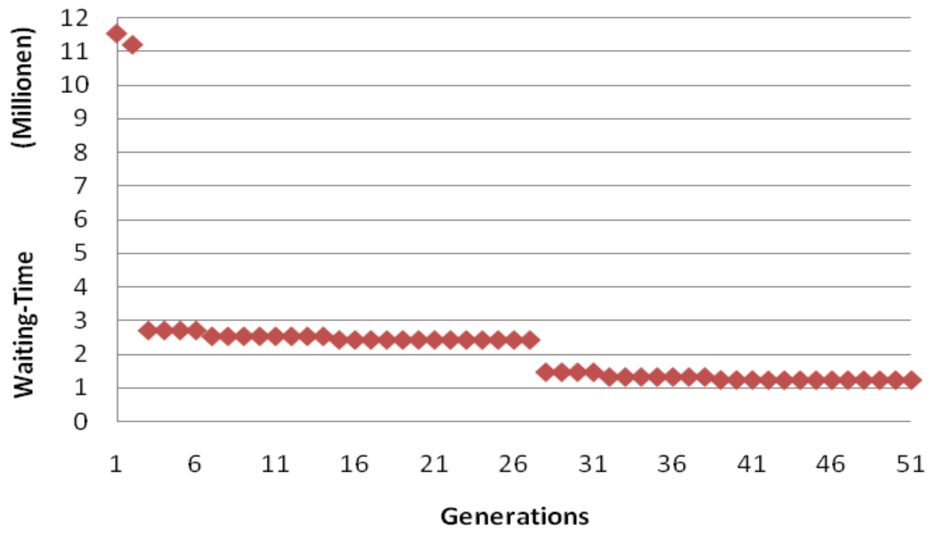


Fig.2.14. Convergence of GA search using tournament selection

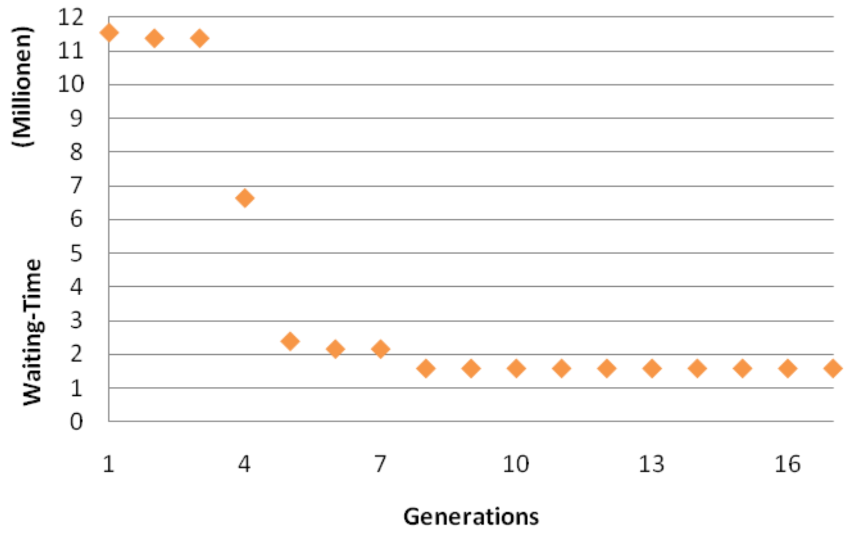


Fig.2.15. Convergence of GA search using ranking selection

One of the maintenance schedule obtained is outlined in Table2.2, where "X" represents the performing of the maintenance. The resulting total traffic delay time is 1238462.240 time units. It is a good result for the binary encoding problem, however, compare to the integer represented approach (shown in the next chapter), the delay time is too long.   Each year only 5 bridges are maintained.

Table2.2   Maintenance Schedule

Bridge Index	Year 1	Year2	Year3	Year 4	Year5
0	--	--	--	--	--
1	--	--	--	--	--
2	--	--	--	--	X
3	--	--	--	--	--

4	--	--	--	--	--
5	--	--	--	--	--
6	--	--	--	--	--
7	--	--	--	--	--
8	--	--	--	--	--
9	--	--	--	--	--
10	--	--	--	--	--
11	--	--	--	--	--
12	X	--	--	--	--
13	--	--	--	--	--
14	--	--	--	--	--
15	--	--	--	--	--
16	--	--	--	--	--
17	--	--	--	--	--
18	--	--	--	X	--
19	--	--	--	--	--
20	--	--	--	--	--
21	--	--	--	--	--
22	--	--	--	--	--
23	--	--	--	--	--
24	--	--	--	--	--
25	--	--	--	--	--
26	--	--	--	--	--
27	--	--	--	--	--
28	--	X	--	--	--
29	--	--	--	--	--
30	--	--	--	--	--
31	--	--	--	--	--
32	--	--	--	X	--
33	--	X	--	--	--
34	--	--	--	--	--
35	--	--	--	--	--
36	--	--	--	--	--
37	--	--	--	--	--
38	--	--	--	--	--
39	--	--	--	--	--
40	--	--	--	--	--
41	--	--	--	--	--
42	--	--	X	--	--
43	--	--	--	--	--
44	--	--	X	--	--
45	--	--	--	--	--
46	--	--	--	--	--
47	--	--	--	--	--
48	X	--	--	--	--
49	--	--	--	--	--
50	--	--	X	--	--
51	--	--	--	--	--
52	--	--	--	--	--
53	--	--	--	--	--
54	--	--	--	--	--
55	--	--	--	--	--
56	--	--	--	--	X
57	--	--	--	--	--
58	--	--	--	--	--
59	--	--	--	--	--
60	--	--	--	--	--
61	--	--	--	--	--
62	--	--	--	--	--
63	--	--	--	--	--
64	X	--	--	--	--
65	--	--	--	--	--
66	--	--	--	--	--
67	--	--	--	--	--
68	--	--	--	--	X
69	--	--	--	--	--

70	--	X	X	X	--
71	--	--	--	--	--
72	--	--	X	--	--
73	--	--	--	--	--
74	--	--	--	--	--
75	X	--	--	--	--
76	--	--	--	--	--
77	--	--	--	X	--
78	--	X	--	--	--
79	--	--	--	--	--
80	--	--	--	--	--
81	--	--	--	--	--
82	--	--	--	--	--
83	--	--	--	--	--
84	--	X	--	--	--
85	--	--	--	--	X
86	--	--	--	--	--
87	--	--	--	--	--
88	--	--	--	--	--
89	--	--	--	--	--
90	--	--	--	--	--
91	X	--	--	--	--
92	--	--	--	--	X
93	--	--	--	--	--
94	--	--	--	--	--
95	--	--	--	--	--
96	--	--	--	--	--
97	--	--	--	X	--
98	--	--	--	--	--
99	--	--	--	--	--

2.5.3 Discussion of the Results

The different parent selection schemes do not show big difference from Fig.2.12, however, they are all convergent as shown in Fig.2.13 - Fig.2.15.
From Table 2.2, we can see that bridge 70 is maintained three times (year 2, 3 and 4) within 5 years as shown in Table2. It is not only unnecessary, but also takes the chances for other bridges away. This is one drawback of the binary represented GA, since the mutation and crossover are totally without control. An approach with a representation by integer-matrix will improve this weakness.

2.5.4 Different Parameter Settings

This testing case uses binary represented GA with one-point crossover (or two-point crossover), bit-flip mutation, and three parent-selection schemes (tournament selection, roulette wheel selection and ranking selection).
First, we keep all other parameters same as before, but increase the population size to 500 and 1000 (instead of 100 used before). Fig.2.16 and 2.17 show the convergence for problem using tournament selection and flipping one random bits mutation operators. Now, we achieve a much better distribution Table 2.3. From this table we can see that bridge 72 is maintained twice within five years, however, other bridges are only maintained once. The respective traffic delay time is

1045965.180 time units. Compared to Table 2.2 (population size is 100, traffic delay time is 1238462.240 time units), the delay time is 16% shorter.

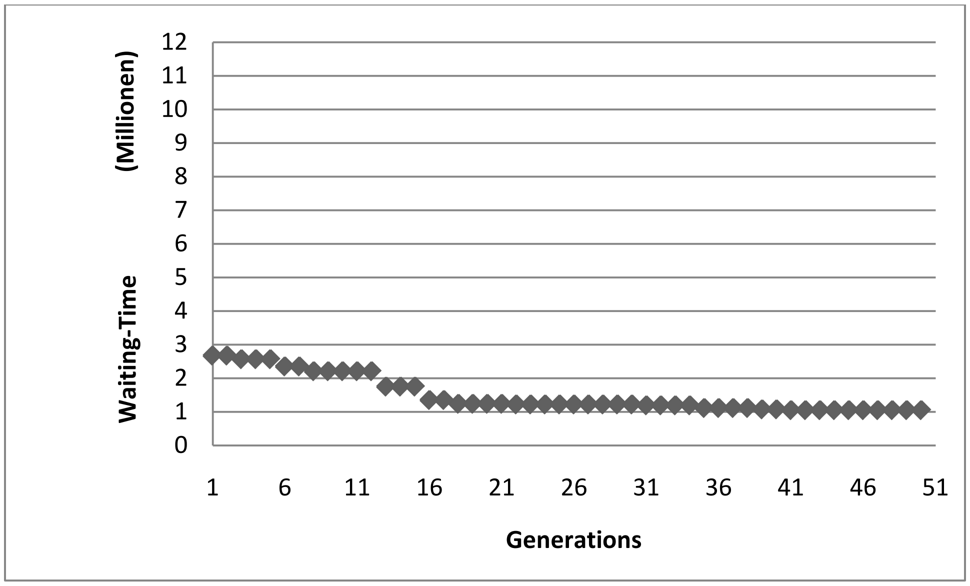


Fig.2.16. Convergence with population size N = 500, tournament selection

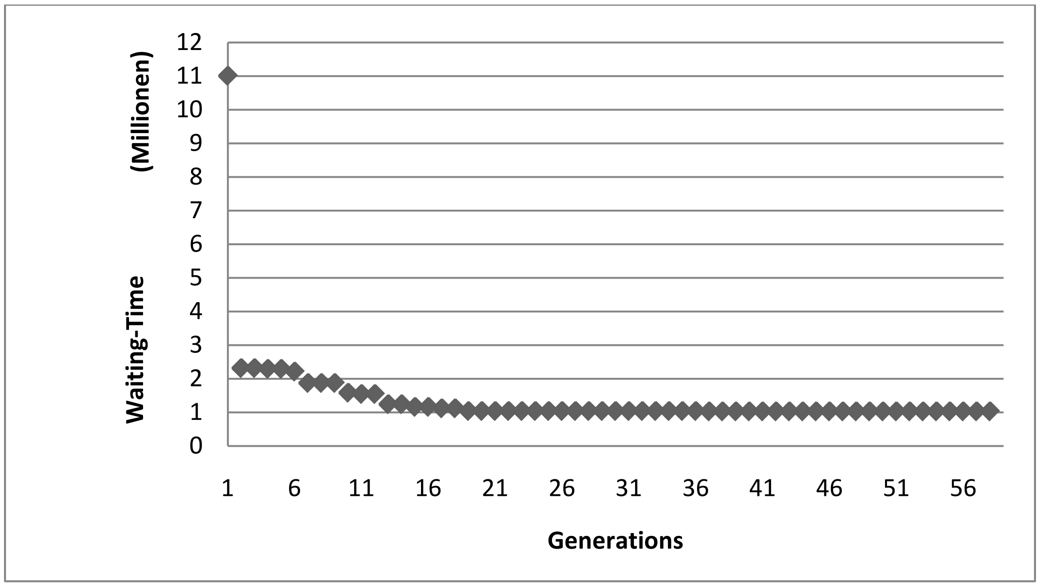


Fig.2.17. Convergence with population size N = 1000, tournament selection

Table2.3 Maintenance Schedule

Bridge Index	Year 1	Year2	Year3	Year 4	Year5
0	--	--	--	--	--
1	--	--	--	--	--
2	--	--	--	--	--
3	--	--	--	--	--
4	--	--	--	--	--
5	--	X	--	--	--

6	--	--	--	--	X
7	--	--	--	--	--
8	--	--	--	--	--
9	--	--	--	--	--
10	--	--	--	--	--
11	--	--	--	--	--
12	--	X	--	--	--
13	--	--	--	--	--
14	--	X	--	--	--
15	--	--	--	--	--
16	--	--	--	--	--
17	--	--	X	--	--
18	--	--	--	--	--
19	--	--	--	--	--
20	--	--	--	--	--
21	--	--	--	--	--
22	--	--	--	--	--
23	--	--	X	--	--
24	--	--	--	--	--
25	--	--	--	--	--
26	--	--	--	--	--
27	--	--	--	--	--
28	--	--	--	--	--
29	--	--	--	--	--
30	--	--	--	--	--
31	--	--	--	--	--
32	--	--	--	--	--
33	X	--	--	--	--
34	--	--	--	--	--
35	--	--	--	--	--
36	--	--	--	--	--
37	--	--	--	--	--
38	--	--	--	--	--
39	--	--	--	--	--
40	--	--	--	--	--
41	--	--	--	--	--
42	--	--	X	--	--
43	--	--	--	--	--
44	--	--	--	--	--
45	--	--	--	--	--
46	--	--	--	--	--
47	--	--	--	--	--
48	--	X	--	--	--
49	X	--	--	--	--
50	--	--	--	X	--
51	--	--	--	--	--
52	--	--	X	--	--
53	--	--	--	--	X
54	--	--	--	--	--
55	--	--	--	--	--
56	--	--	--	--	--
57	--	--	--	--	--
58	--	--	--	X	--
59	--	--	--	--	--
60	--	--	--	--	--
61	--	--	--	--	--
62	--	--	--	--	--
63	X	--	--	--	--
64	--	--	--	--	--
65	--	--	--	--	--
66	--	--	--	--	--
67	--	--	--	--	--
68	--	--	--	--	X
69	--	--	--	--	--
70	X	--	--	--	--
71	--	--	--	--	--

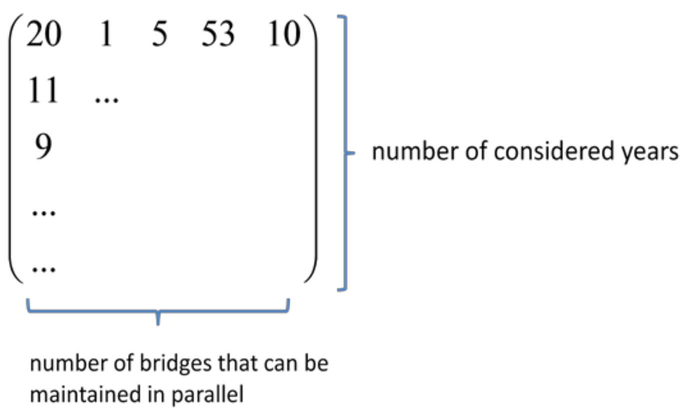
72	X	--	--	X	--
73	--	--	--	--	--
74	--	--	--	--	--
75	--	--	--	--	--
76	--	--	--	--	--
77	--	--	--	--	--
78	--	--	--	--	--
79	--	--	--	--	--
80	--	--	--	--	--
81	--	--	--	--	--
82	--	--	--	--	--
83	--	--	--	--	--
84	--	X	--	--	--
85	--	--	--	--	--
86	--	--	--	X	--
87	--	--	--	--	--
88	--	--	--	--	--
89	--	--	--	X	--
90	--	--	--	--	X
91	--	--	--	--	X
92	--	--	--	--	--
93	--	--	--	--	--
94	--	--	--	--	--
95	--	--	--	--	--
96	--	--	--	--	--
97	--	--	X	--	--
98	--	--	--	--	--
99	--	--	--	--	--

Part 2 Integer Representations

Binary representations are not always the most suitable if the gene can take one of a set of values [6]. An integer encoding is probably more suitable than a binary encoding when designing the variation operators.

2.6 Chromosome Representation and Initialization

The chromosome is represented in the form of integer matrix M of size of M_y X M_w, for M_y = 5 and M_w = 5, where the elements of M are the index number of the bridges, My is the number of considered years and M_w is the number of bridges that can be maintained in parallel. The matrix will look like following:


20 1 5 53 10
11 ...
9
...
...
number of considered years
number of bridges that can be maintained in parallel

Initially, the matrix should not have repeat elements.

2.7 Crossover

We use special crossover strategy for the maintenance problem in 2.9.1: taking R_1 rows from one parent and R_2 rows from the other. As a result, the offspring schedule will have the plans from both parents according to the year. For example, in Fig.2.18 child1 has the 2nd year plan from DAD and 1st, 3rd to 5th year plan from MUM. The crossover procedure is shown in Fig.2.19.

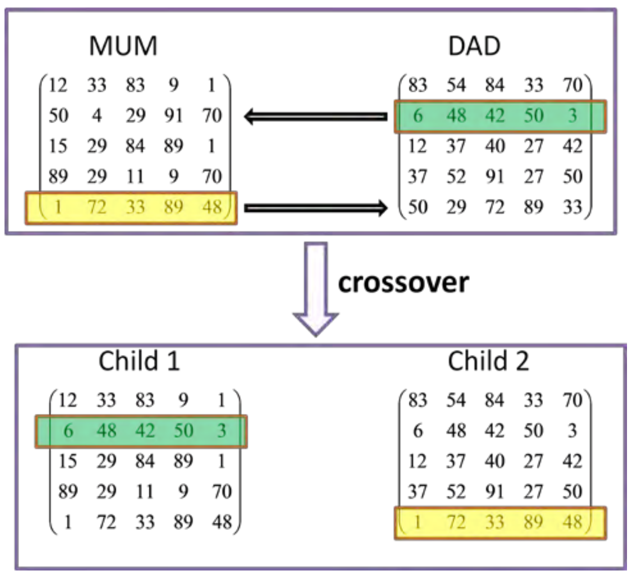


Fig.2.18. Crossover (R=1)

```
BEGIN
Pick a random row R1 from MUM;
Pick a random row R2 from DAD;
IF ( R1 = R2 ) DO
Exchange the row R1 and R2 between MUM and DAD;
ELSE
Substitute row R1 from MUM  for  row R1 from DAD;
Substitute row R2 from DAD  for  row R2 from MUM;
OD
END
```

Fig.2.19. Pseudo code for crossover ($R_1 = R_2 = 1$)

2.8 Mutation

For integer-matrix represented maintenance GA, we use two principal forms of mutation, both of which mutate each chromosome independently.

2.8.1 Random Resetting

Fig.2.20 illustrates changing one random entry in the integer-matrix to a random integer value from the set of permissible values.

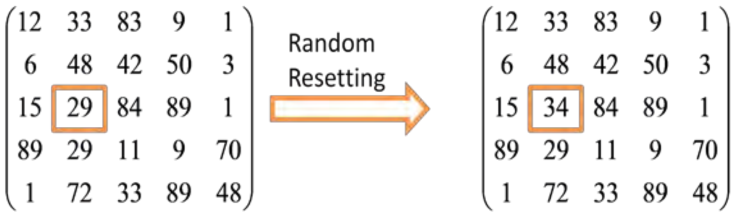

$$\begin{pmatrix} 12 & 33 & 83 & 9 & 1 \\ 6 & 48 & 42 & 50 & 3 \\ 15 & \boxed{29} & 84 & 89 & 1 \\ 89 & 29 & 11 & 9 & 70 \\ 1 & 72 & 33 & 89 & 48 \end{pmatrix} \xrightarrow{\text{Random Resetting}} \begin{pmatrix} 12 & 33 & 83 & 9 & 1 \\ 6 & 48 & 42 & 50 & 3 \\ 15 & \boxed{34} & 84 & 89 & 1 \\ 89 & 29 & 11 & 9 & 70 \\ 1 & 72 & 33 & 89 & 48 \end{pmatrix}$$

Fig.2.20. Random Resetting one element

2.8.2 Substitution

Substitution is the process of first picking two random entries from two random rows and then exchanging their values. It is shown in Fig.2.21.

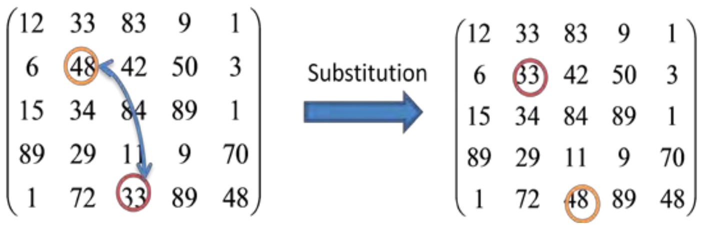


Fig.2.21. Substitution

2.9 Simulation Results

2.9.1 Test Problem

The problem is almost the same like the binary representation GA sample problem (2.5.1), however, cost constraint (Fig.2.22.) is added for the Integer-Matrix represented problem. From this figure, we can see that the maintenance cost increases as the bridge condition index gets bigger. The Problem is listed in Table 2.4.

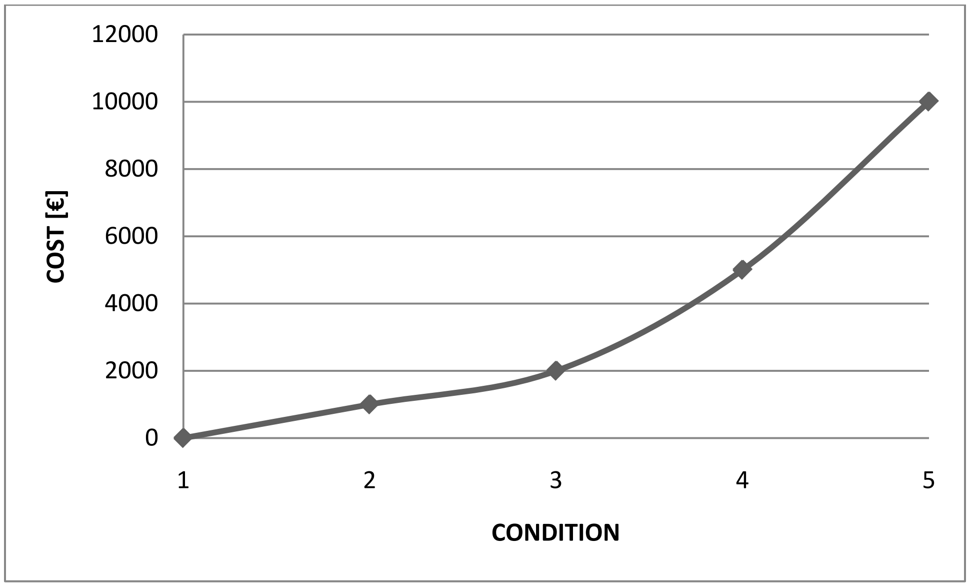


Fig.2.22. Maintenance Cost for different Conditions

Table 2.4 Test Problem

Objective Functions (Fitness Functions)	min f(x)=$\max_{i=0,..,4}(\sum_{j=0}^{j=5} gene[i][j] \times waitingtime(\,gene[i][j])$
Constrains	1. condition[j] < 6 j=0,..,99; 2. Only 5 bridges are under maintenance each year. 3. Budget ≤ 35,000 € / year

Termination Conditions	solutions keep unchanged (σ ≤ 0.0001) in 10 successive generation.

2.9.2 Performance Measures

Like the binary-represented GA, there are three schemes of parent selection: roulette wheel selection, tournament selection, ranking selection. Population size = 100, probability of mutation = 0.002, and test run = 20. However, the length of chromosome is different as binary-represented GA, we set it to 25 (since the integer-matrix is 5x5, i.e., maintenance schedule has a five-year plan which maintains 5 bridges each year). The simulation results are shown in Fig.2.23. Similar as binary represented GA test (2.5.1), roulette wheel selection performs worse than the other two selection schemes.

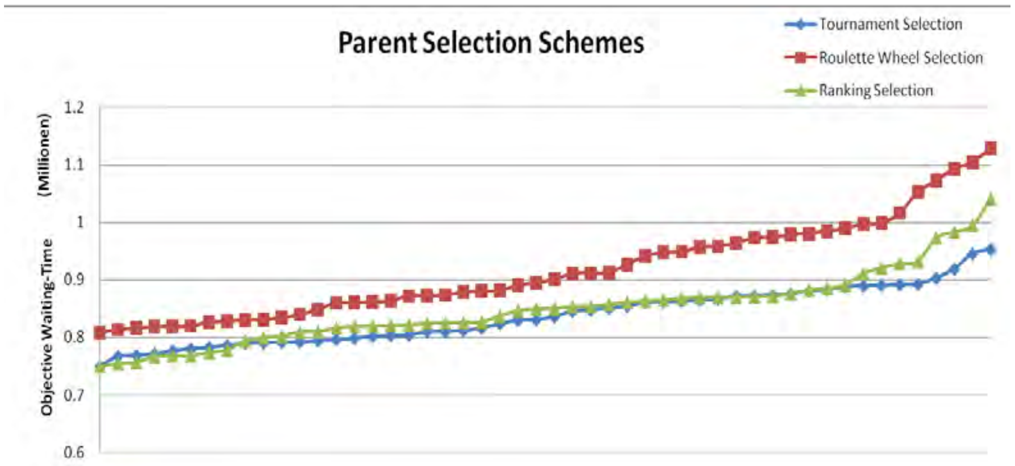


Fig.2.23. Different Parent Selection Schemes for Integer-Matrix Represented GA

Fig.2.24-Fig.2.26 shows the convergence of different parent selection schemes. Fig.2.24 shows the convergence of tournament selection, Fig.2.25 shows it for roulette wheel selection, and Fig.2.26 shows for ranking selection. From these three figures , we find that they all converges very fast (around 2 or 3 generation).

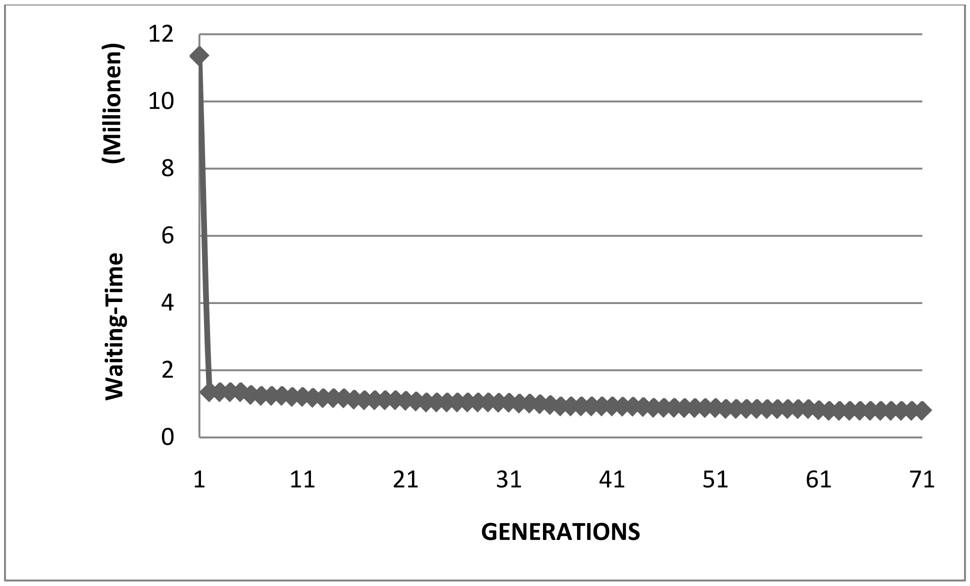


Fig.2.24. Convergence of GA search using tournament selection

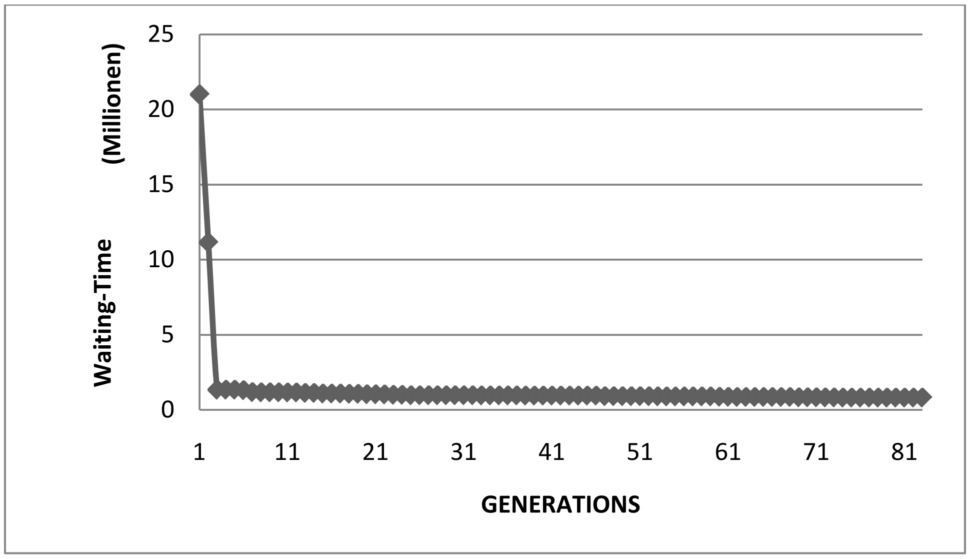


Fig.2.25. Convergence of GA search using roulette wheel selection

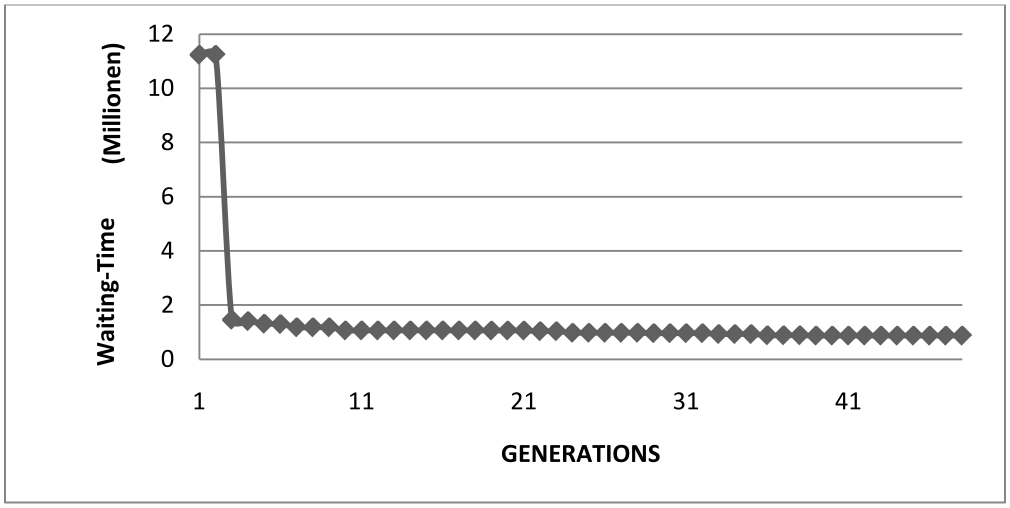


Fig.2.26. Convergence of GA search using Ranking selection

One of the maintenance schedule obtained is outlined in Table2.5, where the matrix members represent the bridges which will perform maintenance. The resulting total traffic delay time is 785522.950 time units, which is much shorter than the case using binary representation. Each year only 5 bridges are maintained. In this schedule bridge 1 and 54 are maintained twice, which is unnecessary.

Table 2.5 Maintenance Schedule

YEAR	BRIDGES MAINTANENCE					COST
year 1	84	1	70	50	42	22000
year 2	63	37	33	19	12	21000
year 3	46	90	84	89	91	13000
year 4	82	54	18	9	33	10000
year 5	40	54	1	42	72	12000

2.9.3 Discussion of the Results

When we constraint the problem to exact 5 bridges maintained per year, the integer represented GA processes much better fitness value (traffic delay time) than binary represented GA. Fig.2.27 is the difference between integer and binary represented GA with tournament selection, population size = 100. We can see that integer represented problem converges much faster than binary represented one, and it also achieves shorter traffic delay time.

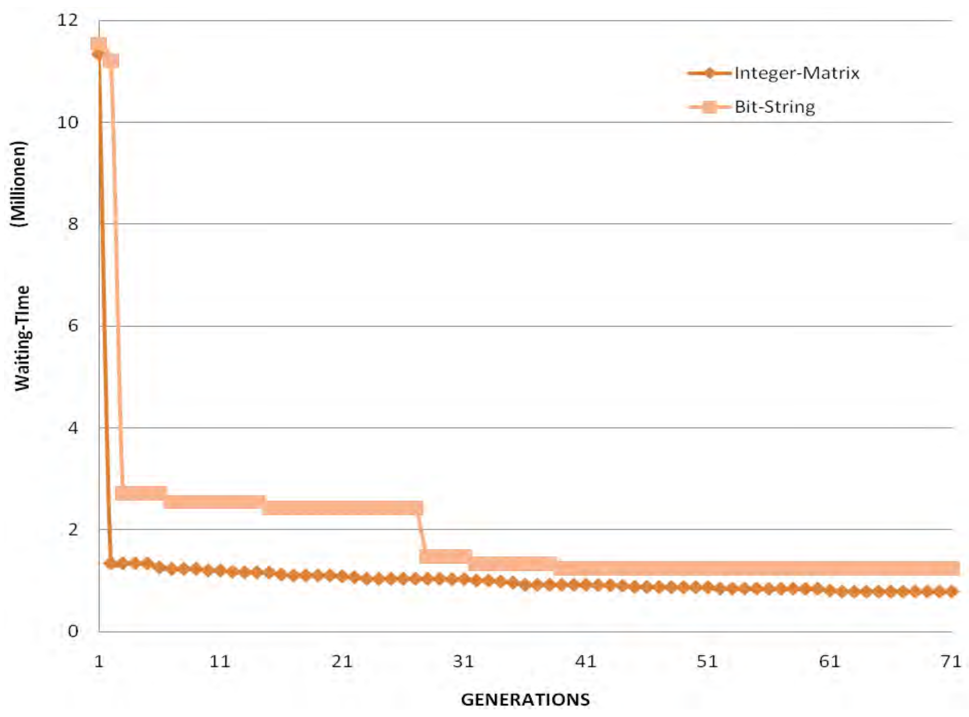


Fig.2.27. Integer-Matrix represented GA VS. Binary represented GA

2.9.4 Different Parameter Settings

We keep the all other parameters same as before, but increase the population size to 500 (instead of 100 used before). Fig.2.28 shows the convergence for problem using tournament selection. Now, we achieve a new schedule (with fitness value = 845522.860 time units) in Table2.6. The traffic delay time is longer, compared to the setting with 100 population.

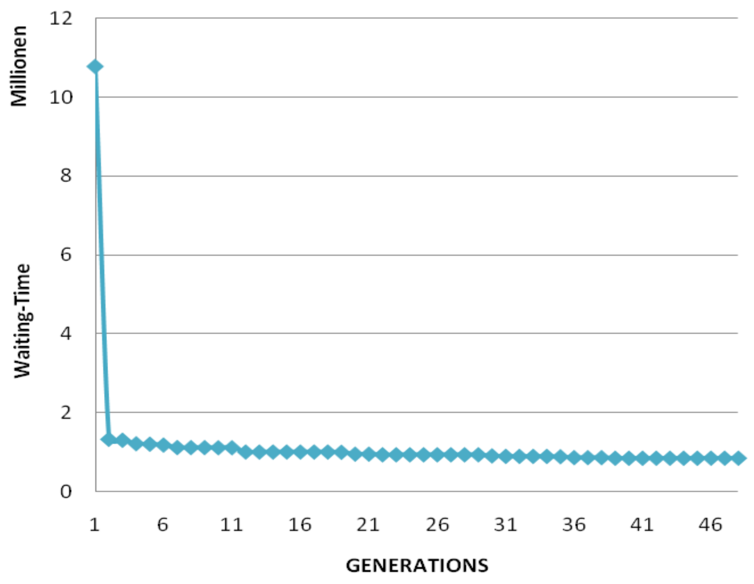


Fig.2.28. Convergence with population size N = 500, tournament selection

Table 2.6 Maintenance Schedule

YEAR	BRIDGES MAINTANENCE					COST
year 1	53	42	86	84	29	17000
year 2	70	29	62	33	1	15000
year 3	22	38	83	37	12	23000
year 4	89	46	33	50	48	12000
year 5	53	30	72	38	37	20000

Chapter 3 Multi-Objective Genetic Algorithms (MOGA)

3.1 From SOGA to MOGA

The single objective formulation is extended to reflect the nature of multi-objective problems where there is not one objective function to optimize, but many. Thus, there is not one unique solution but a set of solutions. This set of solutions is found through the use of Pareto Optimality Theory [7] [8].
Besides having multiple objectives, there are a number of fundamental differences between single objective and multi-objective optimization, as follows [9]:
•Two or more goals instead of one;
•Dealing with two or more search spaces.

3.2 Pareto Terminology

When having more than one objective function, the notion of "optimum" is changed into finding good compromises or "trade-offs" among all the objective functions like in Fig.3.2. The notion of "optimum" that is most commonly adopted is that originally proposed by Francis Ysidro Edgeworth in 1881. The notion of optimum in multi-objective optimization was later generalized by Vilfredo Pareto (in 1896).

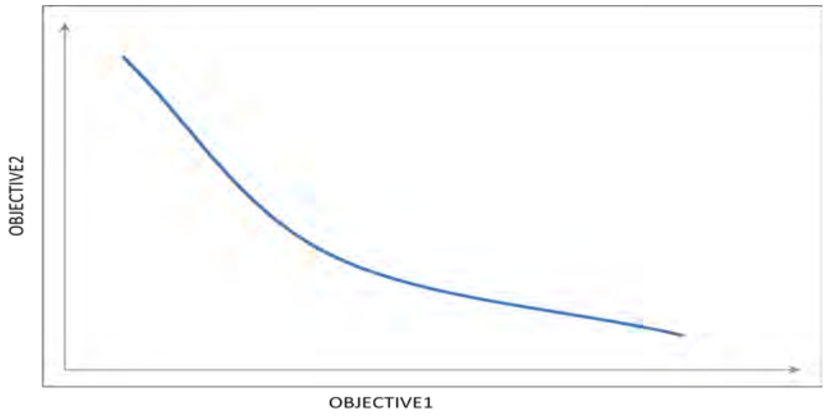


Fig.3.2 An example of a problem with two objective functions: the trade-off surface is delineated by a curved line.

The multi-objective optimization problem has been proposed by a lot of studies, here quotes the definition form Andrzej Jaszkiewicz [16]:
The general multi-objective combinatory optimization (MOCO) problem is formulated as:

$$\min\left\{ f_1(x) = z_1, \ldots, f_J(x) = z_J \right\}$$

s.t. $x \in D$

where: solution $x = [x_1, ..., x_I]$ is a vector of discrete decision variables, D is the set of feasible solutions.

The image of a solution x in the objective space is a point $z^x = [z_1^x, ... z_J^x]$, such that

$z_j^x = f_j(x), j = 1, ..., J$.

A point $z^1 \in Z$ dominates $z^2 \in Z$, $z^1 \succ z^2$, if $\forall j z_j^1 \leq z_j^2$ and $z_j^1 < z_j^2$ for at least one j .

Solution x^1 dominates x^2, $x^1 \succ x^2$, if the image of x^1 dominates the image of x^2 (Pareto Dominace). A solution $x \in D$ is efficient (Pareto-optimal) if there is no $x' \in D$ such that $x' \succ x$.
Point being image of an efficient solution is called non-dominated. The set of all efficient solutions is called efficient set and denoted by N. The image of the efficient space in the objective space is called non-dominated set[16].
The plot of the objective functions whose non-dominated vectors are in the Pareto optimal set is called the Pareto front. All solutions on the Pareto front are optimal.

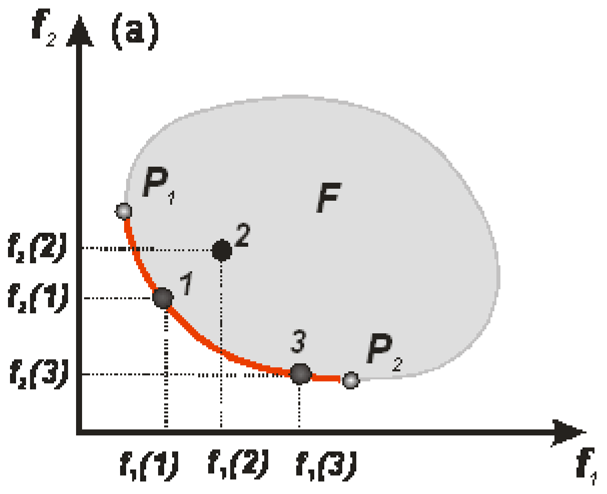


Fig.3.3 Example of bi-objective optimum

Fig.3.3 shows an example of minimum in a bi-objective space (f_1, f_2).The Pareto front is the boundary between the points *P_1* and *P_2* of the feasible set *F.* Solutions 1 and 3 are non-dominated Pareto optimal solutions. Solution 2 is not Pareto optimal as solution 1 has simultaneously smaller values for both objectives. Therefore the aim of multi-objective optimization is to obtain a representative set of non-dominated solutions.

3.3 Various MOGAs

3.3.1 Non-dominated Sorting Genetic Algorithm – NSGA

N. Srinivas and K. Deb [10] proposed the non-dominated sorting GA (NSGA) in 1995. The NSGA algorithm is based on several layers of classifications of the individuals [14]. At first, sort the individuals on the basis of the non-domination: all the non-dominated individuals are categorized into one class, with a dummy fitness value. To maintain the diversity of the population, these classified individuals are shared with their dummy fitness values [14]. Then ignore these classified individuals, rank the left individuals of the group and then repeat this process until all the individuals are classified. Then select individuals based on these classes. The pseudo code for NSGA is given in Fig.3.1.

```
BEGIN
     INITIALISE  population;
     EVALUATE objective functions and constraints;
     ASSIGN rank based on Pareto Dominance [14];
     Compute niche distance;
     Assign shared fitness $\sigma_{share}$;
      REPEAT UNTIL( TERMINATION CONDITION is satisfied) DO
         PARENT SELECTION;
         CROSSOVER with the setting crossover rate;
         MUTATE the offspring;
         EVALUATE  each individual;
         ASSIGN rank based on Pareto Dominance ;
         Compute niche distance;
          Assign shared fitness;
       OD
END
```

Fig.3.1 Pseudo Code for the NSGA algorithm

In K. Deb's later paper [11] he mentioned the main criticism of the NSGA approach, which is showed below:

• The non-dominated sorting has the computational complexity of O (MN^3) (where M is the number of the objective functions and N is the population size), which is the same as other optimal approach.

•Lack of elitism [11]: elitism scheme keeps a trace of the current fittest member and makes sure it is always kept in the population. Recent research ([12] [13]) shows that the elitism can preserve good solutions, and also can speed up the convergence of the Pareto Front.

•Need for specifying the sharing parameter σ_{share} [11] which influents the diversity in a population significantly.

3.3.2 An Improved Version of the NSGA Algorithm – NSGAII

K. Deb et al. [11] [15] proposed an improved version of the NSGA algorithm – NSGAII in 2002. In [14] it mentioned that this algorithm is currently used in most multi objective evolutionary algorithm (MOEA) comparisons. The detail algorithm will be explained in 3.4. Here we'll only give some general idea about the NSGAII:

• NSGAII works with a faster non-dominated sorting algorithm, which requires at most O (MN^2) (where M is the number of the objective functions and N is the population size) computations.

• Use density estimation and crowded comparison operator to preserve diversity of the population.

3.4 A Fast and Elitist Multi-Objective Genetic Algorithm: NSGAII

The following presents the NSGAII using a fast non-dominated sorting procedure, an elitist-preserving approach, and a parameter-less niching operator [15].

3.4.1 A Fast Non-dominated Sorting Approach

The currently-used non-dominated sorting algorithm has a computational complexity of O(MN^3),where M is the number of objectives and N is the population size. When sorting the population, each individual has to be compared with every other individual in the population to find out whether it is dominated, that needs O(MN) comparisons for each individual and O(MN^2) comparisons for all individuals to find out the first Pareto front. Then discount the individual of the first front and repeat the procedure until all the individuals are ranked. The whole process without any book-keeping has the computational complexity of O(MN^3).

The NSGAII uses a better book-keeping strategy when sorting the individuals. The first individual is kept in the first Pareto front, the second individual is only compared with the first one, and the third individual is compared with the first two individuals (only if both are part of the Pareto front), and so on. This requires a maximum of O(N^2) domain checks. During the comparison, if the individual p (the one that will enter the Pareto front P' temporarily) dominates any individual q in the Pareto front P', then q is deleted from the Pareto front P'. And if p is dominated by any member in the Pareto front P', p is ignored. If p is not dominated by any member of the Pareto front P', p enters this Pareto front P'. And that is how the Pareto front grows with non-dominated solutions. The algorithm is shown below in Fig.3.4, where P is the set of all individuals.

```
P' = Pareto-Front(P)
BEGIN
    P'={1}
    For each $p \in P \wedge p \notin P'$
      $P' = P' \cup p$
      For each $q \in P' \wedge q \neq p$
            If $q \prec p$, DO
                P'=P'\{q}
            Else if $p \prec q$, DO
                    P'=P'\{p}
            OD
END
```

Fig.3.4.Pseudo Code for Finding the Pareto front

To find other fronts, the member from P' will be neglected temporarily and the above searching procedure goes on. At the end of these comparisons, solutions of the first Pareto front are stored in F_1, solutions of the second Pareto front are stored in F_2 , and so on. The whole procedure is shown in Fig.3.5.

```
F=fast-non-dominated-sorting(P)
BEGIN
    $i = 1$
    WHILE $P \neq \varnothing$ DO
            $F_i$ = Pareto-Front(P)
            P = P\$F_i$
            $i{+}{+}$
    OD
END
```

Fig.3.5. Procedure of fast non-dominated sorting

3.4.2 Density Estimation Metric

The original NSGA uses sharing function approach (see 3.3.1) to maintain the solutions spreading out, while the sharing functions are set by users. There are two difficulties with this sharing function approach [15]:

•The performance of the sharing function method in maintaining a spread of solutions largely depends on the chosen of sharing function.

•The complexity is $O(N^2)$, since each solution must be compared with all other solutions in the population.

The NSGAII uses an concept called density estimation to replace any user-defined parameter. Also, the new approach has a better computational complexity according to [15].
To get an estimate of the density of solutions surrounding a particular solution in the population, we calculate the average distance of two points on each objective direction. The crowding distance $i_{distance}$ is the size of the largest cubic enclosing only the point i, like shown in Fig.3.6.

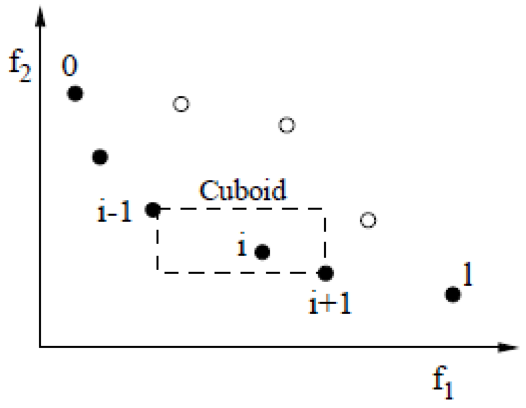


Fig.3.6 The crowding distance is shown (compare to [15]).

Calculation of the crowding distance $i_{distance}$ needs sorting the population according to each objective function. The procedure is shown below in Fig.3.7, where I is a non-dominated set, and $\mathrm{I}[i].m$ is the m-th objective function value of the i-th individual in the set I.

```
Crowding-Distance($\mathrm{I}$)
BEGIN
  $l$ = the length of $\mathrm{I}$
  For each $i$, set $\mathrm{I}[i]_{distance}$ =0          //initialize distance
  For each objective m
      $\mathrm{I} = sort(\mathrm{I}, m)$                    //sort using each objective value
      $\mathrm{I}[1]_{distance} = \mathrm{I}[l]_{distance} = \infty$      //the boundary solutions are assigned an infinite value
      For $i$ =2 to $(l-1)$
          $\mathrm{I}[i]_{distance} += (\mathrm{I}[i+1].m - \mathrm{I}[i-1].m)$
END
```

Fig.3.7 the algorithm for the crowding distance calculation (compare to [15])

According to [15], the above algorithm has the computational complexity of $O(MN \log N)$.

3.4.3 **Crowded Comparison Operator** $(\prec_n)$

After all members are assigned a distance metric, we can compare the solutions. Every solution has two attributes:

•non-dominated rank (i_{rank})

•crowding distance $(i_{distance})$

A solution with smaller rank is better. Otherwise, if both solutions have the same rank, then the one with bigger crowding distance (less crowded) is better. Here we choose the symbol from $(\prec_n)$ [15] to implement the comparison of two solutions, as shown in Fig.3.8.

```
IF $(i_{rank} = j_{rank})$ DO
    IF $(i_{distance} > j_{distance})$ DO
        $i \prec_n j$
    OD
ELSE IF $(i_{rank} < j_{rank})$ DO
        $i \prec_n j$
OD
```

Fig.3.8. the definition of operator $\prec_n$

3.4.4 Conclusion

The above three features make the NSGAII working more efficiently. The fast non-dominated sorting algorithm sorts the combination of current generation and the previous one according to the non-domination, and then the density estimation metric calculates the crowding distance of all members in the combination, at last the crowded comparison operator assigns a rank to each member, the pseudo code is in Fig.3.9 (compare to [15]), where P_t is parent population, Q_t is children population, and R_t is the combination of parent and children. $F = (F_1, F_2, ...)$ is the set of non-dominated front, N is the size of population, $|P_t|$ and $|F_i|$ are the length of sets P_t and F_i .

```
BEGIN
    $R_t = P_t \cup Q_t$
    F= fast-non-dominated-sorting( $R_t$ )
    Set $P_{t+1} = \varnothing$ and $i = 1$
    WHILE $(|P_{t+1}| + |F_i| \leq N)$ DO
```

```
            Crowding-Distance( $F_i$ )
            $P_{t+1} = P_{t+1} \cup F_i$
            $i++$
    OD
    Sort $F_i$ with $\prec_n$
    Choose the first $(N - |P_{t+1}|)$ elements of $F_i$ to fill the population $P_{t+1}$
    Generate $Q_{t+1}$ from $P_{t+1}$ ,using SELECTION, CROSSOVER and MUTATION
    $t++$
END
```

Fig.3.9.The process of NSGAII

The computational complexities of the above operations are:

• fast non-dominated sorting is $O(M(2N)^2)$

• crowding distance assignment is $O(M(2N)\log(2N))$

• $\prec_n$ sorting is $O(2N\log(2N))$

So, the overall complexity is $O(MN^2)$.

3.5 The Main Loop

The overall steps for NSGAII optimization show in Fig.3.10.

```
INITIALIZE population $P_0$
EVALUATE objective functions, constraints, and ranks of $P_0$
For (gen=1 TO MAXGENERATION ) DO
    PARENT _SELECTION using Tournament Selection
    CROSSOVER
    MUTATE
    KEEP the Pareto Front ALIVE
    ADD to new population
    gen++
OD
```

Fig.3.10 the overall procedural of the GA optimization

Initially, a random population P_0 is created randomly which is the same as SOGA. Besides, crossover and mutation operation are also the same as in SOGA using integer representation.

3.5.1 Evaluating

After the population P_0 was created, the next step is evaluating the objective functions and specifies constraints. Then based on the specified constraints, Pareto front is created, i.e. the population P_0 is ranked. The procedure of creating Pareto front is shown in Fig.3.11, where N is the population size, and the Flag is the signal of been ranked or not (Flag of dominated individuals is 0, Flag of an individual which rank not being assigned is 1, Flag of non comparable individuals is 3)

```
RANKING( $P_0$ )
BEGIN
    Initializing the ranks RNK to zero
    Initializing all the Flags to 2
    For (k=0 To N) DO
       For(j=0;J<N) DO
           Break if all the individuals are assigned a rank
       For (j=0 To N) DO
           Set the flag of dominated individuals to 2
       OD
       RNK++
       For (i=0 To N) DO
           Select an individual $Ind[i]$ which rank to be assigned
           For(j= To N) DO
              Select the other individual $Ind[j]$ which has not got a rank
              $compare(Ind[j], Ind[i])$
            OD
         Assign the rank = RNK, Flag=1
         OD
      OD
END

$compare(Ind[j], Ind[i])$
IF ( $Ind[j]$.constraints are infeasible)&&( $Ind[i]$ .constraint are feasible) DO
   Set $Ind[j].Flag = 0$
ELSE IF( $Ind[j]$.constraints are feasible)&&( $Ind[i]$ .constraints are infeasible)
    Set $Ind[i].Flag = 0$
    ELSE IF( both are feasible)
             $sort(Ind[j], Ind[i], \prec_n)$
OD
```

Fig.3.11 Ranking procedure

3.5.2 Parent Selection

Tournament selection scheme is chosen for parent selection. During the selection, the crowded comparison operator $(\prec_n)$ is used to pick the fitter one. It works as below in Fig.3.12, where P_i is the population in the i-th generation.

```
Parent_Selection ( $P_i$ )
Randomly pick two parents $Ind[g_1]$ and $Ind[g_2]$
IF $(Ind[g_1].rank > Ind[g_2].rank)$
    $Ind[g_2] \prec_n Ind[g_1]$
ELSE IF $(Ind[g_1].rank < Ind[g_2].rank)$ DO
     $Ind[g_1] \prec_n Ind[g_2]$
     ELSE IF( $Ind[g_1]$.distance > $Ind[g_2]$.distance)
            $Ind[g_1] \prec_n Ind[g_2]$
          ELSE
            $Ind[g_2] \prec_n Ind[g_1]$
OD
Return the fitter individual
```

Fig.3.12 Parent selection based on $(\prec_n)$

3.5.3 Keep Alive

This step is to make sure that the solutions from the better (lower) ranks are chosen into the next generation. The general process is like in Fig.3.13, and the signal has the same means as in 3.4.4.

```
Keep_Alive $(P_t, Q_t)$
Form the global mating pool $R_t = P_t \cup Q_t$
Find the global ranks $F = (F_1, F_2, ...)$
Fill $P_{t+1}$ according to F
```

Fig.3.13 Keep_Alive Process

First, the global mating pool $R_t = P_t \cup Q_t$ is formed with size 2N. Then the Pareto front of R_t is created by the fast non-dominated sorting algorithm. In the current Pareto front set $F = (F_1, F_2, ...)$, F_1 is the best non-dominated set and is definitely chosen into the new population P_{t+1}. If the size of F_1 is smaller than the population size N, F_2 is chosen, and so on.

3.6 Two-Objective Optimization Test Problem

3.6.1 Test Problem

First we describe the multi-objective problem implemented with NSGAII. The numerical example is based on the data in Fig.3.14-3.16. As we can see, there are 100 bridges totally, and their index starts from 1 to 100. Fig.3.14 shows the waiting time for maintenance each bridge, for example, the bridge 10 has waiting time $1.47 \times 10^5 unit$, i.e., vehicle needs $1.47 \times 10^5 unit$ more time to cross the bridge 10 district when bridge 10 is being maintained. The shortest waiting time is $1.00 \times 10^5 unit$ (bridge 2), and the longest is $3.92 \times 10^5 unit$ (bridge 68).

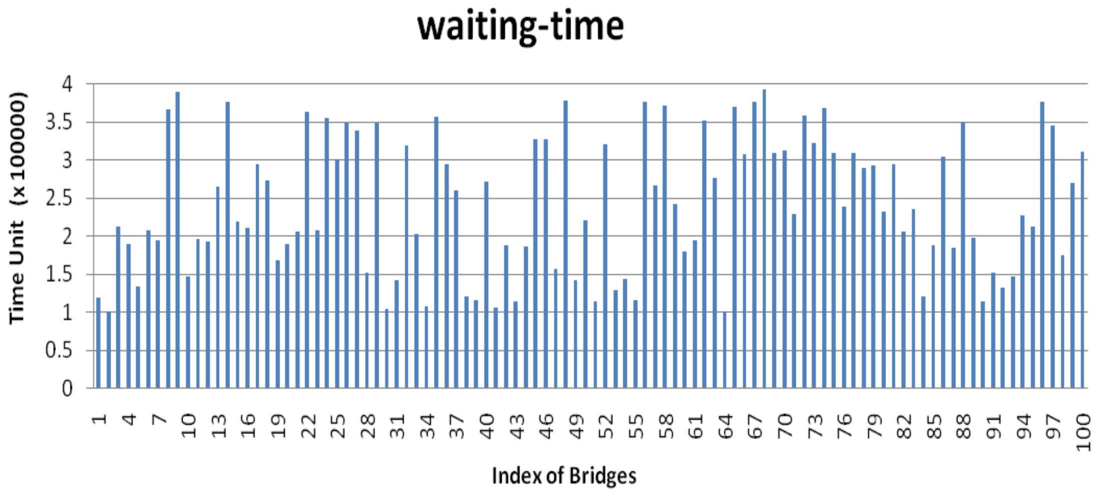


Fig.3.14 Waiting Time for maintenance each bridge

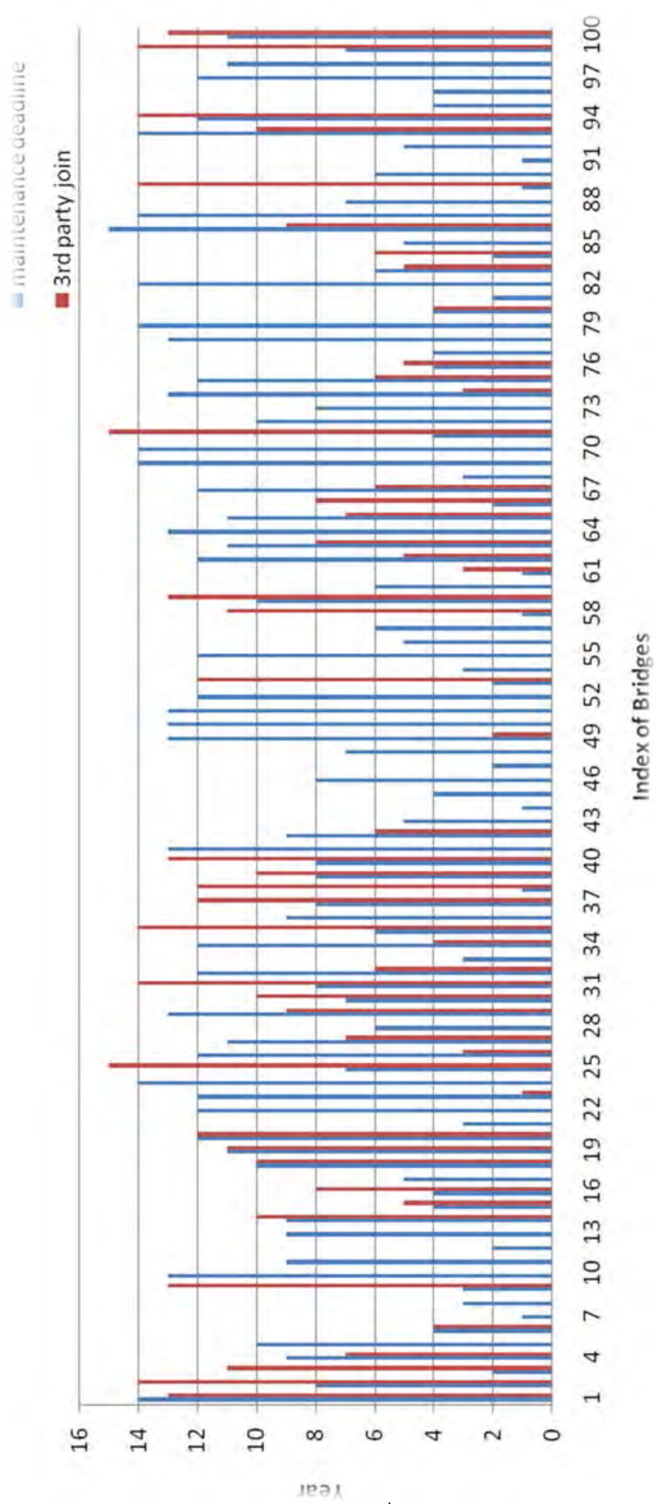


Fig.3.15 Maintenance Deadlines and the 3rd party join time of each Bridge

In Fig.3.15, the blue columns are maintenance deadlines for each bridge, for example, bridge 10 must be maintained before the 3rd year. The earliest deadline is 1 year (bridge 7, 38, 44, 58, 61, 89, 91), and the longest deadline is 15 year (bridge 86). The Red columns in Fig.2.6.2 stand for the time when the 3rd party (other construction companies) will also make construction on the bridges, for example, a 3rd party will make construction on bridge 6 in the 4th year, and there is no extra construction on bridge 10.

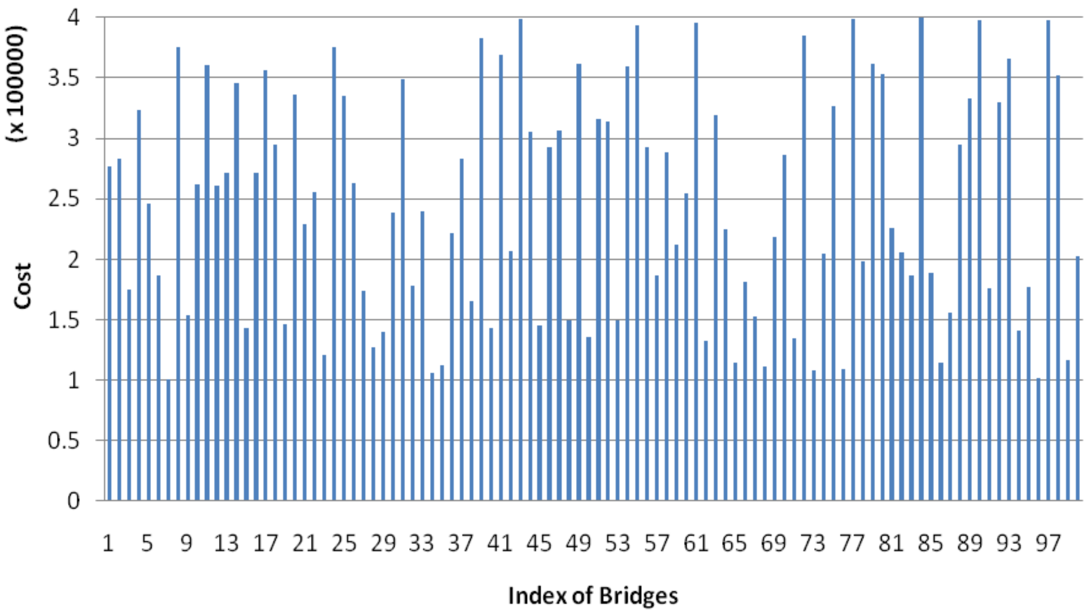


Fig.3.16 Maintenance Fee of each Bridge

Fig. 3.16 shows the maintenance cost of each bridge. The highest fee is 3.96×10^5 (bridge 61), and the lowest is 1.01×10^5 (bridge 7), the average cost is 2.46×10^5.
The problem which we simulate needs a five-year maintenance schedule, and 10 bridges are maintained every year. It has two objective functions with constraints. The problem is described in Table 2.6.1. As the table showing, objective 1 is to minimize the total waiting-time per year, and the 2nd objective tries to do the maintenance in the same year as the 3rd party does the construction, for example, the 3rd party will make construction of bridge 49 at the 2nd year, so it is better to maintain it at the 2nd year. The 1st constraint (will be explained in detail in next part: 3.6.2) makes sure each bridge is maintained (or has already been maintained) before its maintenance deadline. The 2nd and 3rd constraints are to keep the maintenance budget per year between 1.7 million and 2.5 million.

Table 2.6.1 Test Problem with Two Objectives

Objective Functions	$\min \rightarrow f_1 = \max_{i=0}^{4}(\sum_{j=0}^{9} gene[i][j].waiting\_time)$ $\min \rightarrow f_2 = \sum_{i=0}^{4}\sum_{j=0}^{9}(gene[i][j].3rd\_party\_construction\_time - (i+1))$
Constraints	$g_1{++}, if\,(bridge[gene[i][j].deadline = k)$ $g_1 \in [-35, 0]$ $k = 1,...,5$ $i = 0,...,4$ $j = 0,...,9$ $k \le i+1$ $g_2 = 2.5\times10^6 - \sum_{j=0}^{9} gene[i][j].kost, i = 0,...,4$ $g_3 = \sum_{j=0}^{9} gene[i][j].kost - 1.7\times10^6, i = 0,...,4$

All approaches are run for a maximum of 1000 generations and with a population size 100.

3.6.2 Constraint Handling

The principle [15] of constrained NSGAII is that any feasible solution has a better non-dominated rank than any infeasible one. The non-dominated levels of all feasible solutions are given according to objective values of those solutions. For the infeasible solutions, the one has smaller constraint violation has a better rank. Feasible solutions have the constraint values equal to or bigger than 0, and infeasible solutions have the minus constraint values. In Constraint 1(Table 2.6.1), g_1 is set to -35 initially, when a bridge, deadline is k ($k = 1,...,5$), is found in $gene[i][j]$, if $k \le i+1$ (i.e., the maintenance is before the deadline), then g_1 adds 1. If all the bridges (with deadline before 6 year) are maintained before their deadlines, g_1 grows to 0. And if no bridges (with deadline before 6 year) are maintained before their deadlines, g_1 remains -35. So, $g_1 \in [-35, 0]$. But, constraints 2 and 3 have values around 10^6. Then, g_1 multiplied with a penalty value 10^6 to make sure all constraints have the same exponent.

3.6.3 Performance Measures

Fig.3.17 shows the result of the 1000th generation, the blue points are the infeasible solutions, and the red crosses are the feasible solutions. Table 2.6.2 is one of maintenance schedule with feasible solutions. Every year 10 bridges are maintained. The budget per year is between 2.0 million and 2.5 million.

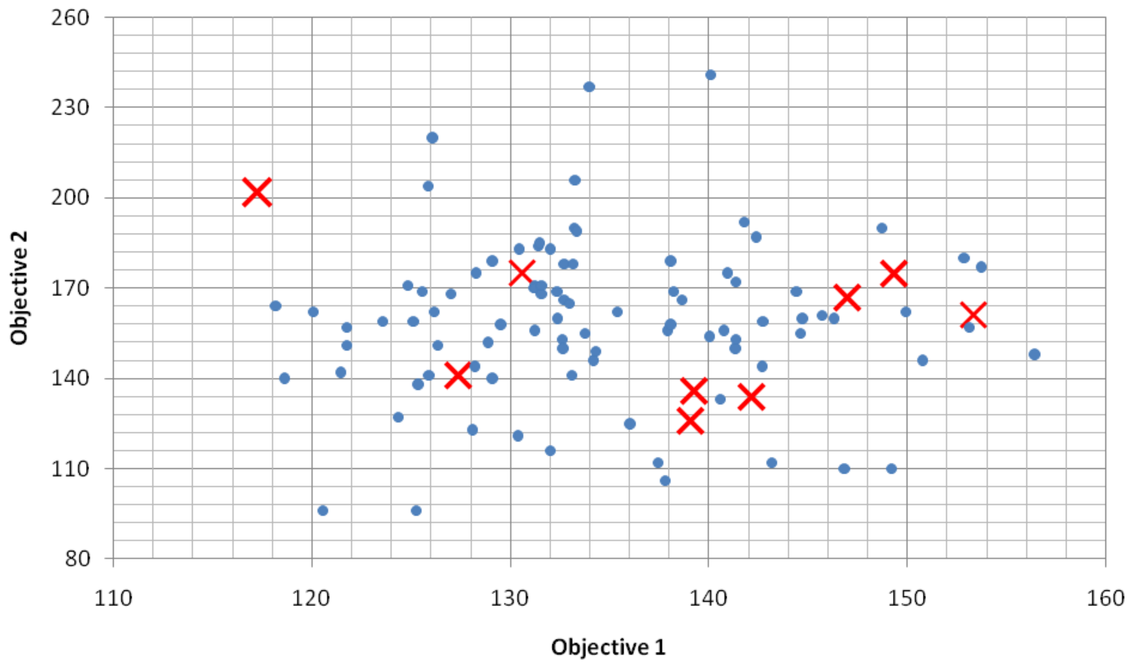


Fig.3.17 NSGAII simulation result

Table 2.6.2 MAINTENANCE SCHEDUEL

1st year	43	6	57	60	88	8	2	37	90	41
2nd year	11	46	83	67	80	5	65	52	75	35
3rd year	94	79	99	14	23	44	53	7	20	32
4th year	48	22	92	95	19	15	70	76	4	0
5th year	55	33	42	84	27	61	81	91	10	16

Fig.3.18-3.20 shows the feasible solutions of generation 252, 676, and 1000. As generation increases, more feasible solutions are found.

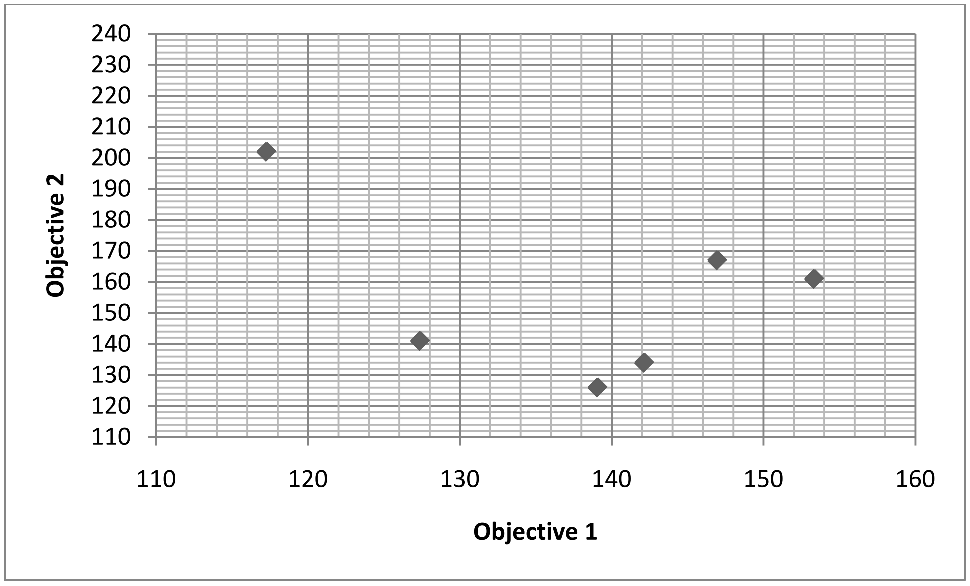


Fig.3.18 Feasible solutions of Generation 252

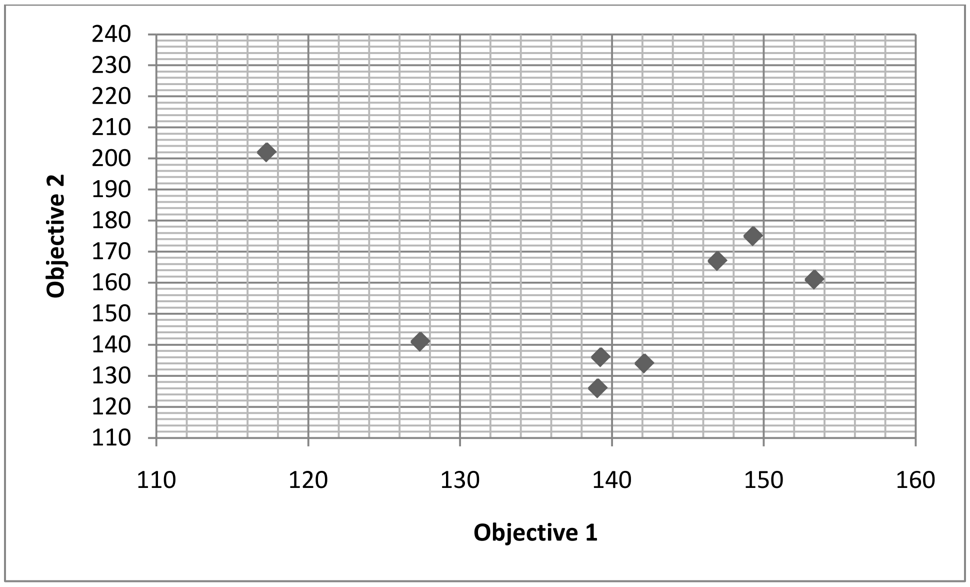


Fig.3.19 Feasible solutions of Generation 676

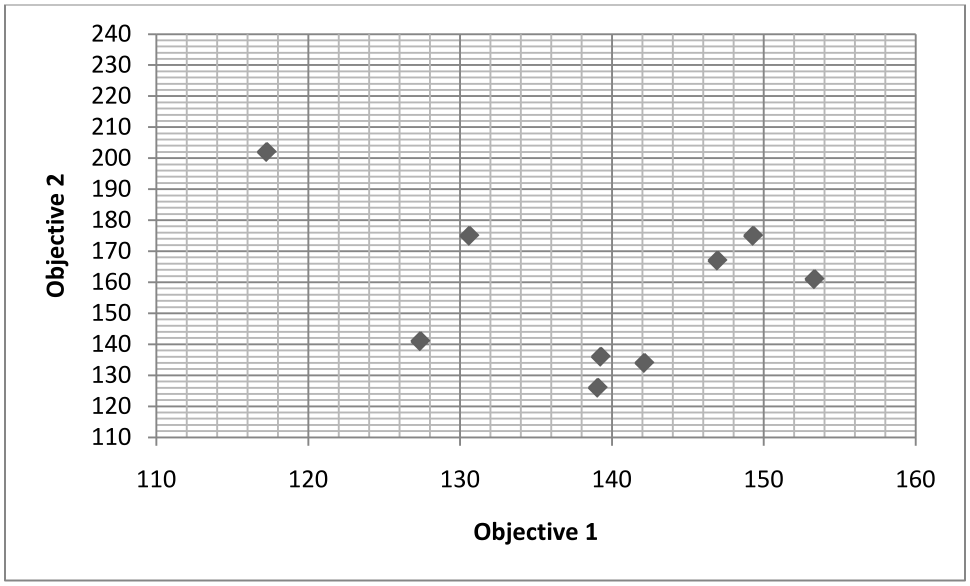


Fig.3.20 Feasible solutions of Generation 1000

3.6.4 Different Parameter Settings

First, we keep all the other parameters same as before, but increase the size of the population to 500 (instead of 100 before).Fig.21 shows all the solutions (feasible and infeasible) in generation 1000, which range from 115 to 165 in objective 1 and from 80 to 250 in objective 2. Fig.3.22 shows only the feasible solutions, which range from 115 to 165 in objective 1 and from 115 to 225 in objective 2. There are totally 15 feasible solutions among 500 solutions. Compared with the simulation running with 100 individual (population size=100), it found more feasible solutions when the population size increases. Table 2.6.3 shows a possible schedule with feasible solution.

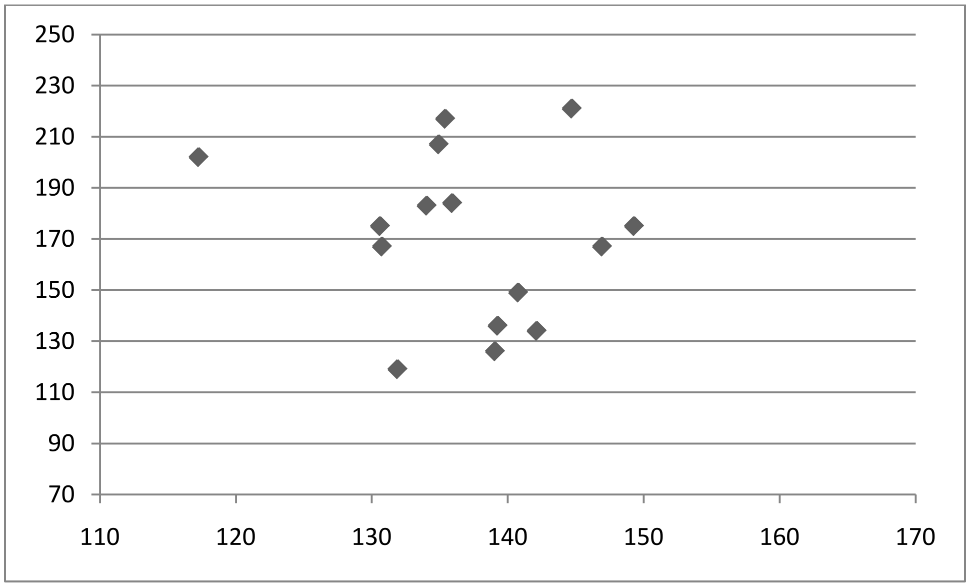


Fig.3.21 All solutions of generation 1000 with population size 500

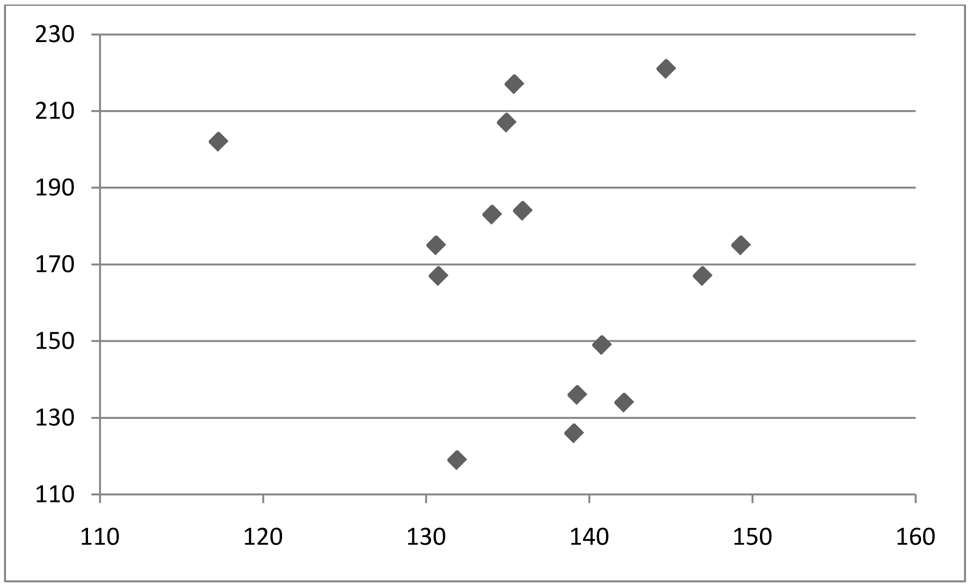


Fig.3.22 Feasible solutions in Generation 1000

Table 2.6.3 MAINTENANCE SCHEDUEL

1st year	90	57	11	6	88	60	44	67	65	43
2nd year	80	52	2	46	8	53	5	83	20	55
3rd year	75	78	53	79	89	76	70	29	32	7
4th year	40	73	42	15	61	9	14	95	94	91
5th year	50	30	28	22	16	84	12	63	86	98

Then, we decrease the size of population to 50 and keep all the other parameter unchanged. Fig.3.23 shows all the solutions in generation 1000. Fig.3.24 shows the feasible solutions in generation 1000. As we can see, it has less feasible solutions compared to the previous two cases (population size =100/ population size= 500). Table 2.6.4 shows a possible maintenance schedule with feasible solution.

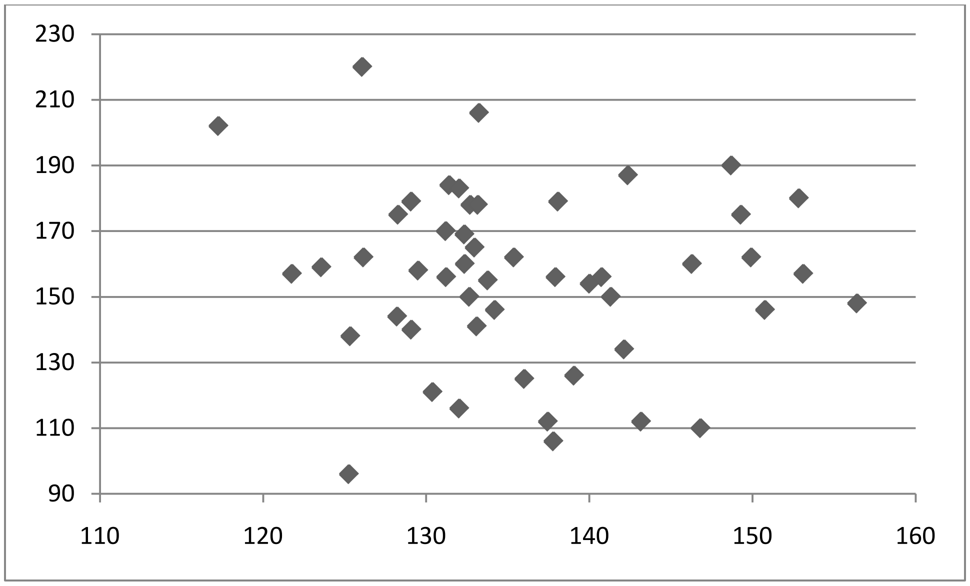


Fig.3.23 All solutions in generation 1000 with population size = 50

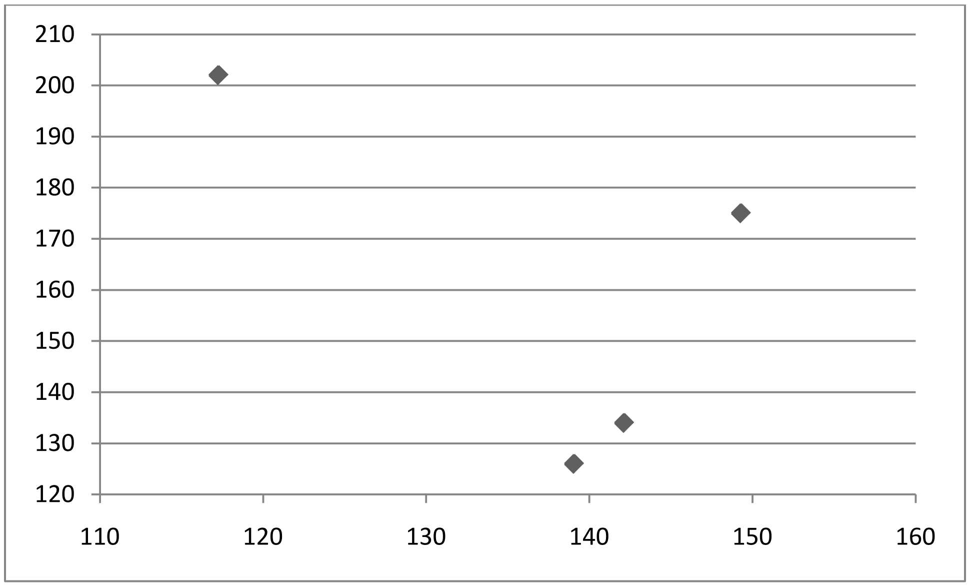


Fig.3.24 Feasible solutions in generation 1000 with population size = 50

Table 2.6.4 MAINTENANCE SCHEDULE

1st year	60	43	85	46	90	57	2	53	37	6
2nd year	32	67	83	44	80	52	15	65	76	11
3rd year	18	5	59	8	7	20	1	79	95	70
4th year	54	84	14	26	91	9	75	4	94	50
5th year	55	48	74	61	64	91	16	27	42	86

3.7 Three-Objective Optimization Test Problem

3.7.1 Test Problem

The numerical example is based on the data from 3.6.1, besides, there is a third objective (group information (Table 2.7.1)) new added. It is better to maintain the bridges in the same group at the same year. For example, maintaining bridge 72 and 78 in the same year is the best solution, while their maintaining year difference being 4 year is the worst solution.

Table 2.7.1 GROUPS

Group 0	Group 1	Group 2	Group 3	Group 4	Group 5
3	29	72	27	52	68
6	31	78	43	59	80
10	41		57	83	
20	44		74	84	
60			86	96	
85			98		

The three-objective problem is described in table 2.7.2. All the other functions are the same as the two-objective problem, except the new added objective function f_3. f_3 works like this: first calculate the year differences between the maintenance years of the bridges belonging to the same group, for example, maintaining bridge 31 at the 1st year and maintaining bridge 41 at the 3rd year, the year difference is 3-1=2. Then add all these year differences in the schedule, we get f_3.

Table 2.7.2 Test Problem with Three Objectives

Objective Functions	$\min \to f_1 = \max_{i=0}^{4}(\sum_{j=0}^{9} fene[i][j].waiting\_time)$ $\min \to f_2 = \sum_{i=0}^{4}\sum_{j=0}^{9}(gene[i][j].3rd\_party\_construction\_time - (i+1))$ $\min \to f_3 = \sum_{gr=0}^{5}\{\lvert i'-i \rvert \mid gene[i][j].group = gene[i'][j'].group = gr; i, i' = 0,\dots,5; j, j' = 0,\dots,9\}$
Constraints	$\min \to f_1 = \max_{i=0}^{4}(\sum_{j=0}^{9} fene[i][j].waiting\_time)$ $\min \to f_2 = \sum_{i=0}^{4}\sum_{j=0}^{9}(gene[i][j].3rd\_party\_construction\_time - (i+1))$ $g_1{+}{+}, if\,(bridge[gene[i][j].deadline = k)$ $g_1 \in [-35, 0]$ $k = 1,\dots,5$ $i = 0,\dots,4$ $j = 0,\dots,9$ $k \le i+1$ $g_2 = 2.5\times10^6 - \sum_{j=0}^{9} gene[i][j].kost, i = 0,\dots,4$ $g_3 = \sum_{j=0}^{9} gene[i][j].kost - 1.7\times10^6, i = 0,\dots,4$

3.7.2 Performance Measures

In this problem, we use a population size of 100, maximum generation of 1000, and run NSGAII. Fig.3.25-3.27 shows all solutions (feasible and infeasible) in generation 10000. From these figures we can see that the lower and upper bounds of the objective function values are: $f_1 \in [117,157]$, $f_2 \in [90,250]$, and $f_3 \in [30,280]$.

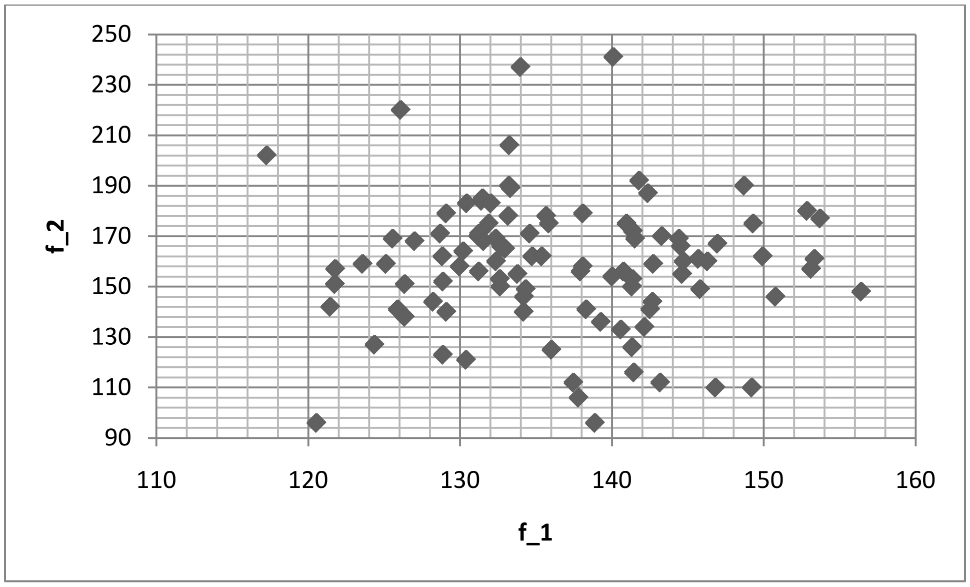


Fig.3.25 $f_1 - f_2$

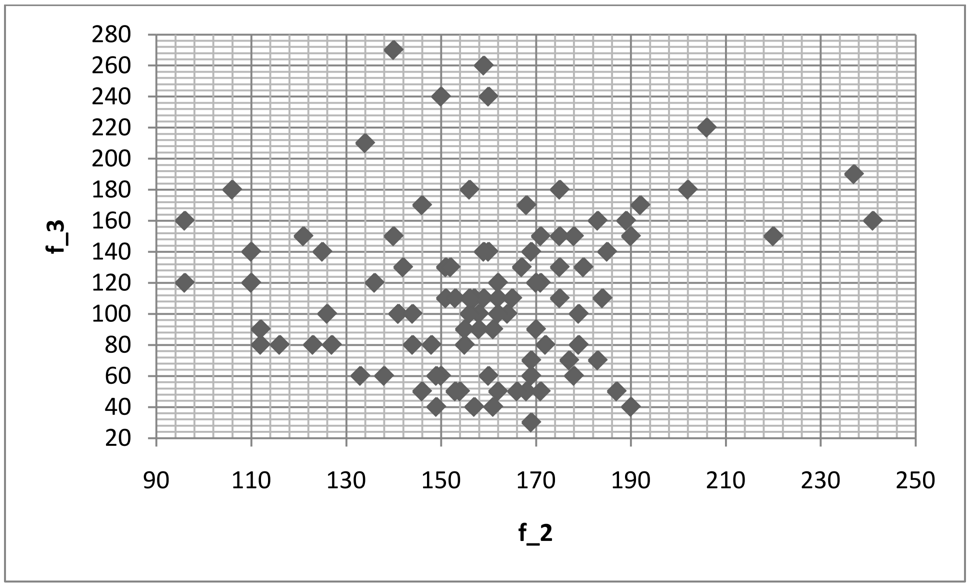


Fig.3.26 $f_2 - f_3$

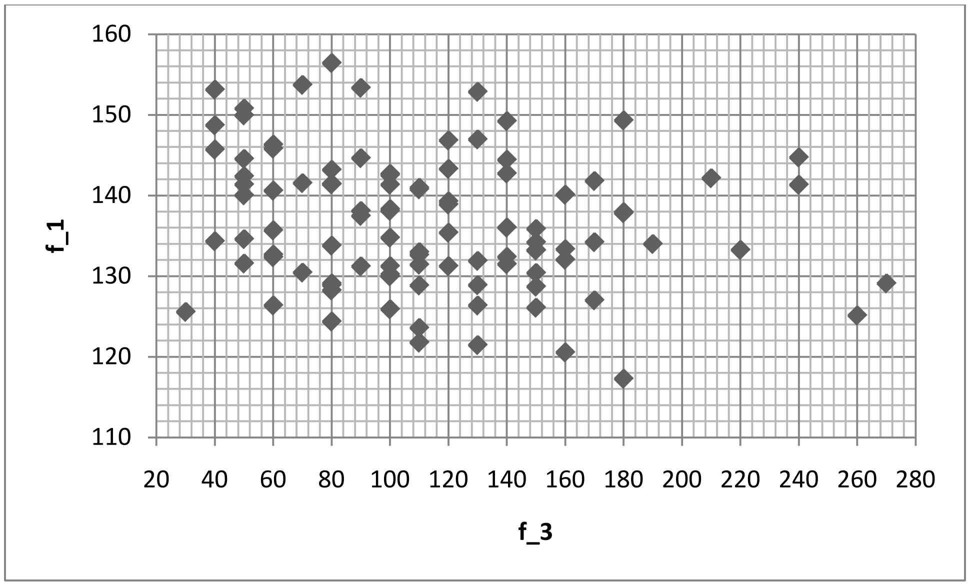


Fig.3.27 $f_3 - f_1$

Fig.3.28-3.30 shows the feasible solutions in generation 1000. We get 9 feasible solutions from 100 solutions. Fig.3.28 shows the solutions in objective 1 and 2, it has the same pattern as two objective feasible solutions (Fig.2.6.6). The range of the objective function values are: $f_1 \in [117,154]$, $f_2 \in [130,205]$, and $f_3 \in [90,210]$.

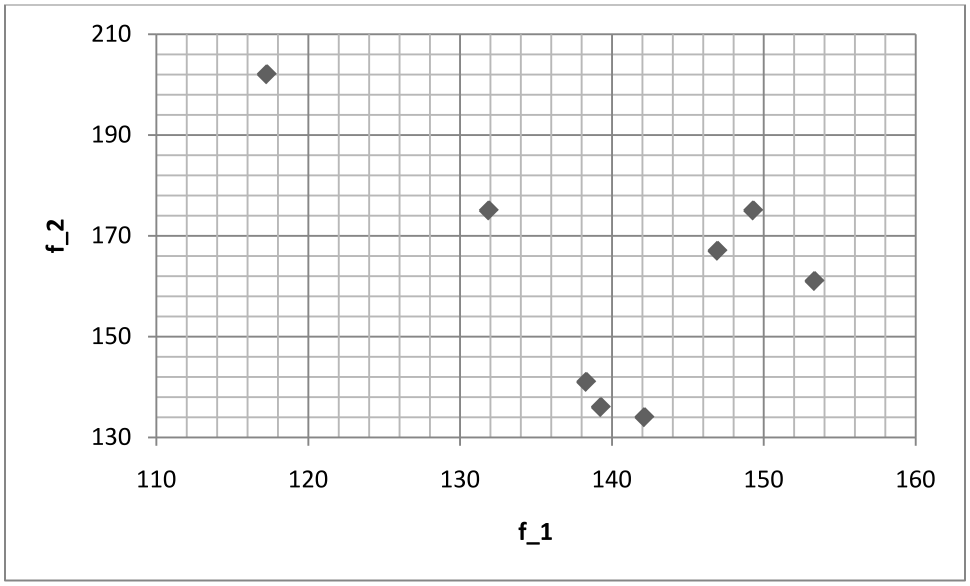


Fig.3.28 $f_1 - f_2$

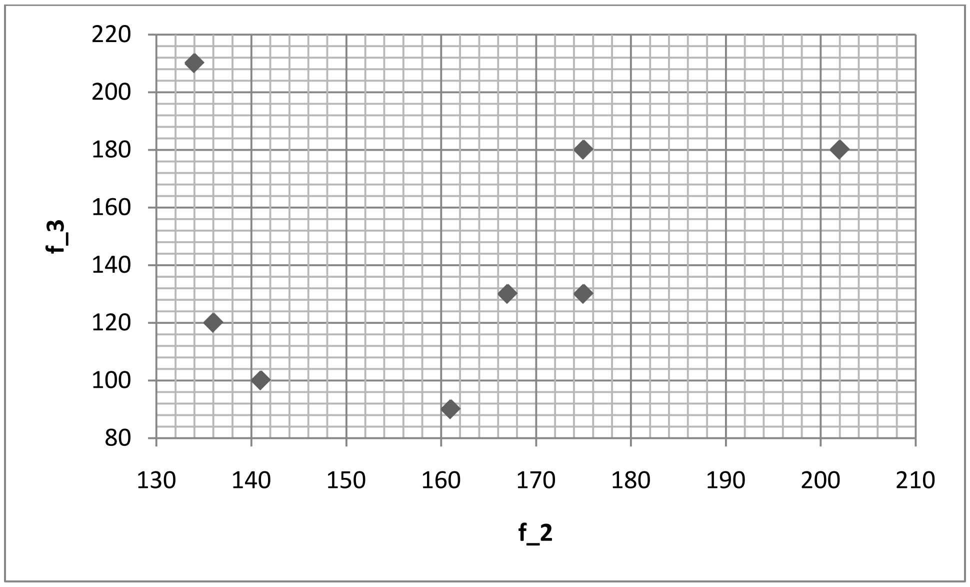


Fig.3.29 $f_2 - f_3$

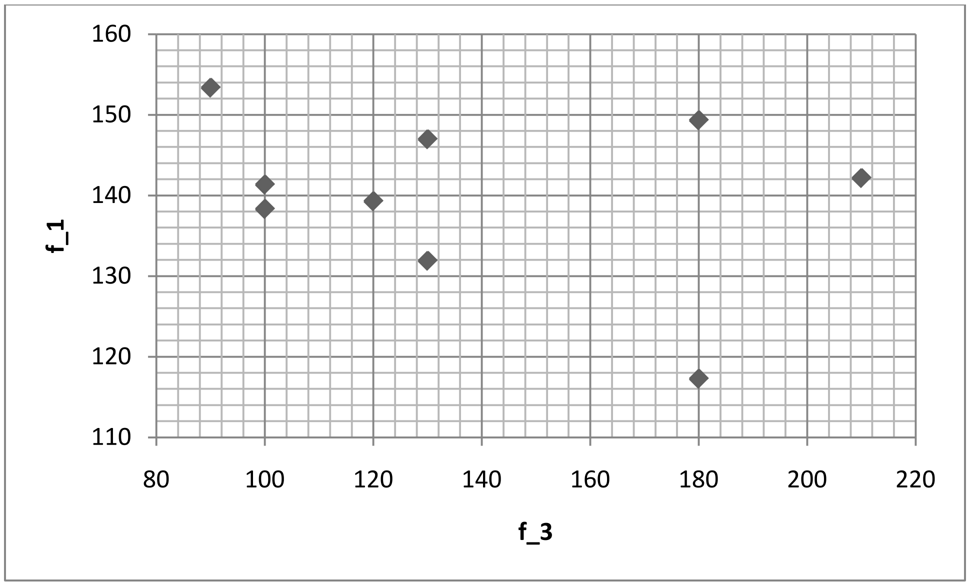


Fig.3.30 $f_3 - f_1$

Table 2.7.3 shows a possible maintenance schedule with three-objective functions. Table 2.7.4 lists the bridges whose maintenance deadlines are 1st year, 2nd year, 3rd year, 4th year, and 5th year. These bridges are then marked in Table 2.7.3, and bridges having the same maintenance deadlines are marked with the same color. For example, bridges 83, 80, 46, 11, 52, 2, 65 are marked with green, and they all have the deadline till the 2nd year. From these two tables, we can see that almost all the bridges are maintained before their deadlines, except bridge 6.

Table 2.7.3 MAINTENANCE SCHEDULE

1st year	57	83	88	80	46	11	60	43	90	37
2nd year	42	52	2	8	32	65	7	16	55	89
3rd year	35	68	20	75	39	76	53	67	30	14
4th year	79	84	15	42	5	70	95	94	44	47
5th year	18	85	21	45	91	25	86	92	50	4

Table 2.7.4 Bridges whose deadline is earlier than 5 year

Deadline	**Bridge Index**									
1st year	6	37	43	57	60	88	90			
2nd year	2	11	46	52	65	80	83			
3rd year	7	8	20	32	53	67				
4th year	5	14	15	44	70	75	76	79	94	95
5th year	16	42	55	84	91					

3.7.3 Different Parameter Settings

We keep all the other parameters same as before, but increase the size of population to 500. Fig.3.31-3.33 shows the feasible solutions in generation 1000. In total there are 26 feasible solutions in 500 solutions.

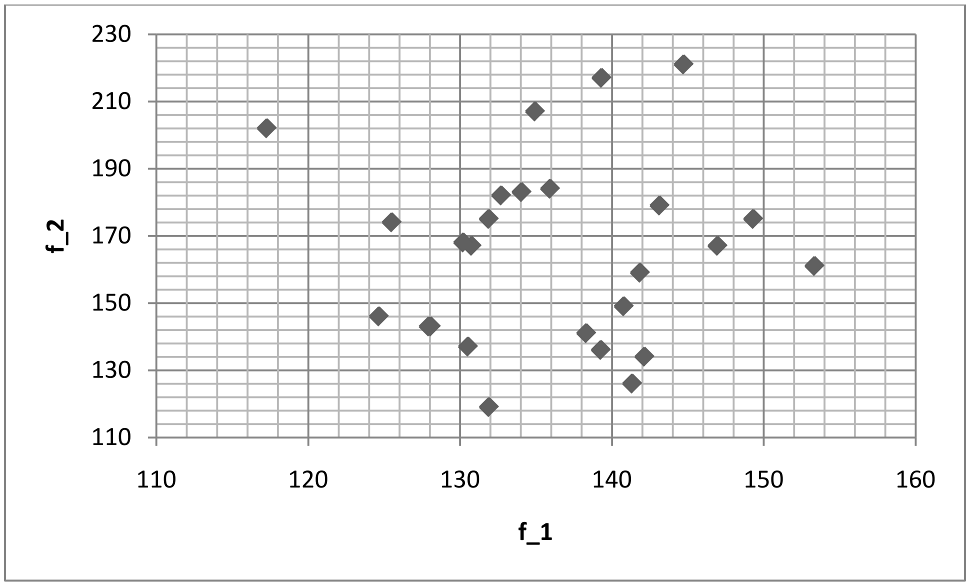


Fig.3.31 $f_1 - f_2$

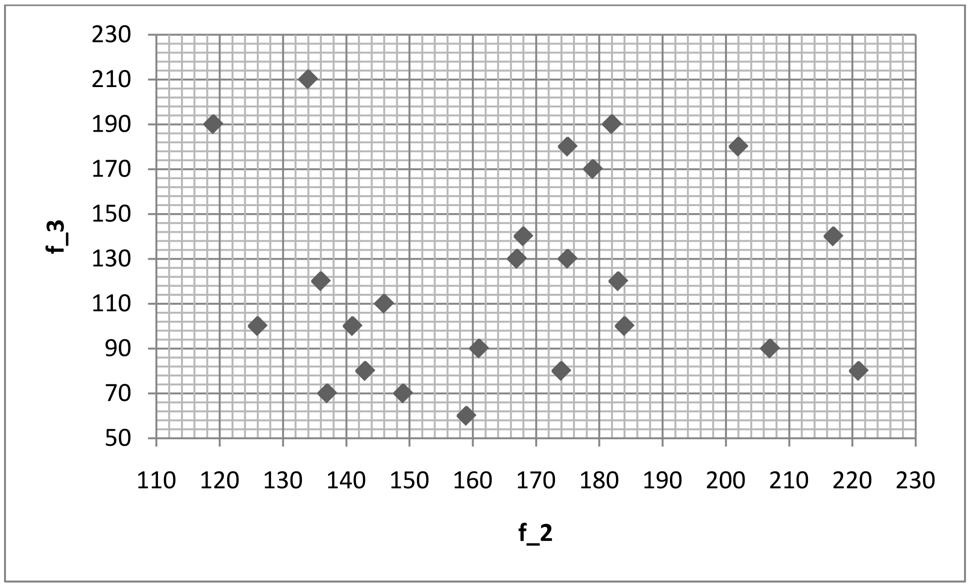


Fig.3.32 $f_2 - f_3$

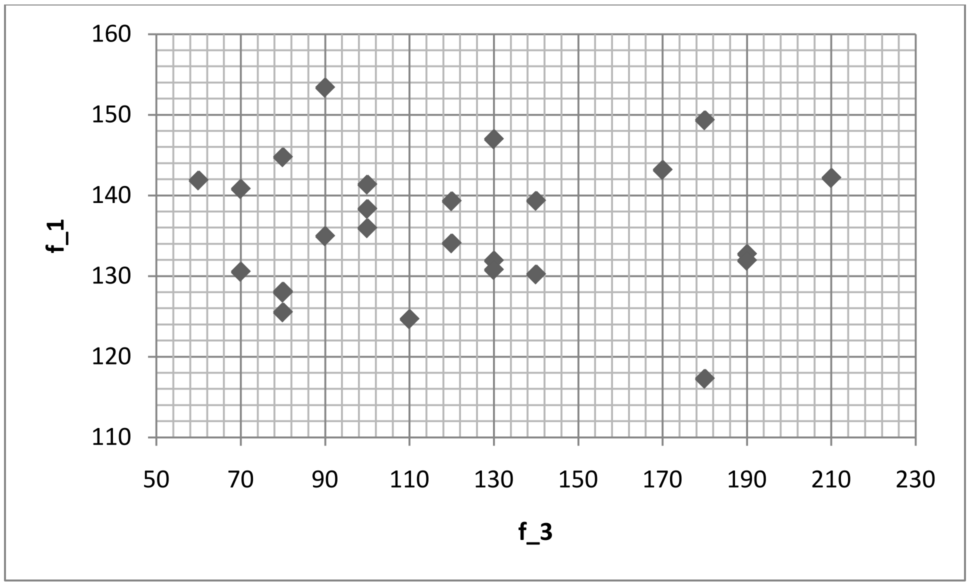


Fig.3.33 $f_3 - f_1$

Table 2.7.5 shows a possible solution. Here the color has the same meaning as Table 2.7.3. From this table, we can see that all the bridges are maintained before their deadlines.

Table 2.7.5 MAINTENANCE SCHEDULE

1st year	37	65	88	6	90	57	60	43	2	46
2nd year	53	80	20	11	7	67	52	95	83	31
3rd year	30	8	84	15	32	44	76	62	75	79
4th year	5	42	64	39	16	66	70	94	14	33
5th year	29	21	55	77	47	9	49	24	93	91

Table 2.7.6 shows the distribution of the working group. Compared to Table 2.7.1, all bridges in group 4 are distributed among the 1st, 3rd, and 4th year, two bridges from group 0 are distributed in the 2nd and the 3rd maintenance year, two bridges from group 1 are separated in the 1st and the 5th year, four bridges from group 3 are distributed among the 2nd, 4th, and 5th year, only one bridges from group 2 are maintained in this schedule, and no bridges from group 5 are maintained.

Table 2.7.6 Group Distribution

maintenance year	**groups**									
1st year	--	--	--	--	4	--	1	--	--	--
2nd year	--	--	3	0	--	--	--	--	--	--
3rd year	--	--	4	--	0	4	2	--	--	--
4th year	4	--	4	--	--	3	--	--	--	--
5th year	--	1	3	--	--	--	--	3	--	--

Table 2.7.7 shows the 3rd-party construction time of the maintenance bridges. For example, bridges 65, 6, 2 are maintained in the 1st year, but the 3rd-party will make constructions of these three bridges in 13 years later. From these tables, we observe that the compromise with 3rd-party participating time is not good.

Table 2.7.7 3rd-party participating information

maintenance year	3rd-party participating time									
1st year	--	14	--	14	--	--	--	--	14	--
2nd year	10	--	--	11	13	10	--	15	--	13
3rd year	15	--	6	5	11	--	--	--	--	13
4th year	--	2	13	--	6	3	14	--	8	--
5th year	--	9	9	5	2	6	13	7	--	--

Then still keep the other parameter unchanged, but decrease the population size to 50. The feasible solutions are showed in Fig.3.34-3.36. There are totally 4 feasible solutions within 50 solutions ingeneration 1000. Table 2.7.8 is a possible schedule with feasible solution.

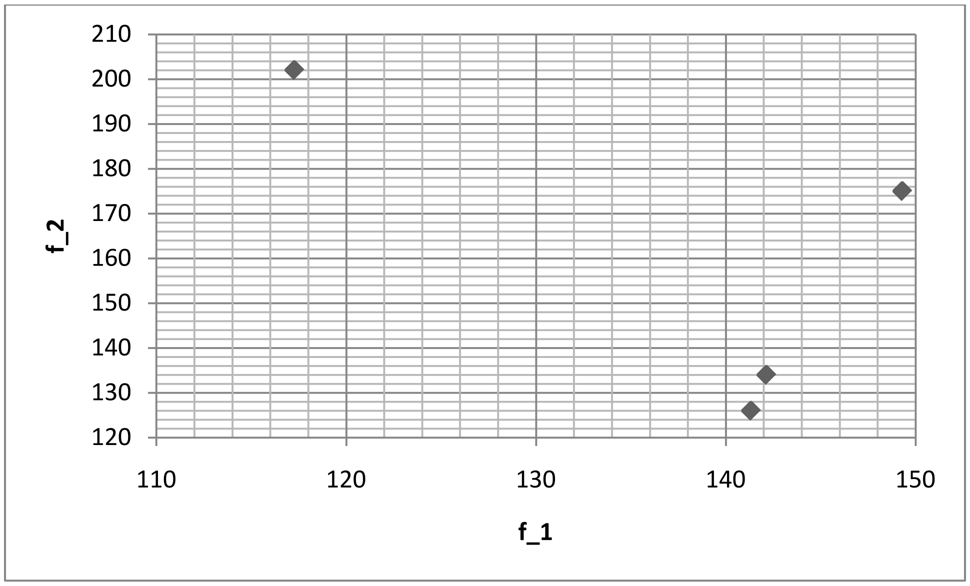


Fig.3.34 $f_1 - f_2$

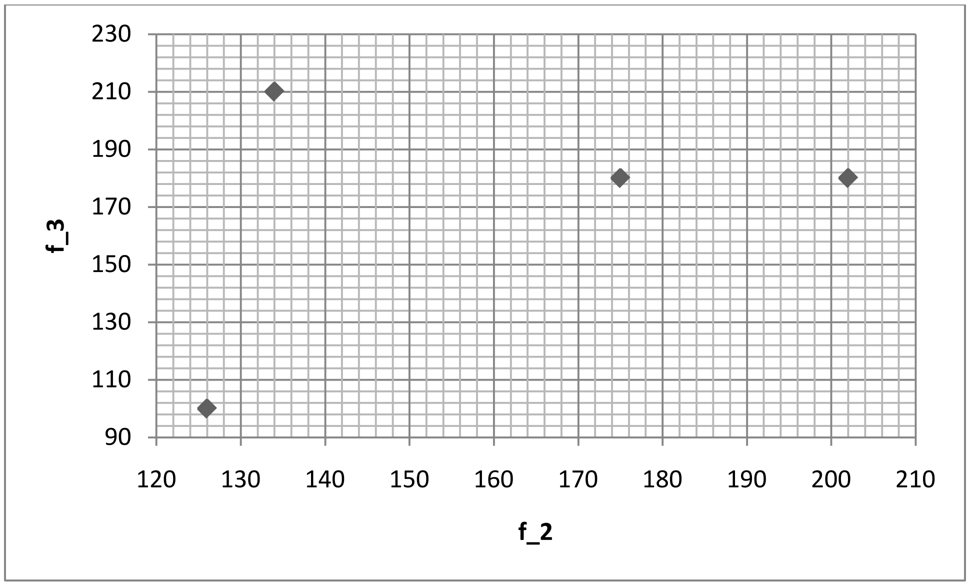


Fig.3.35 $f_2 - f_3$

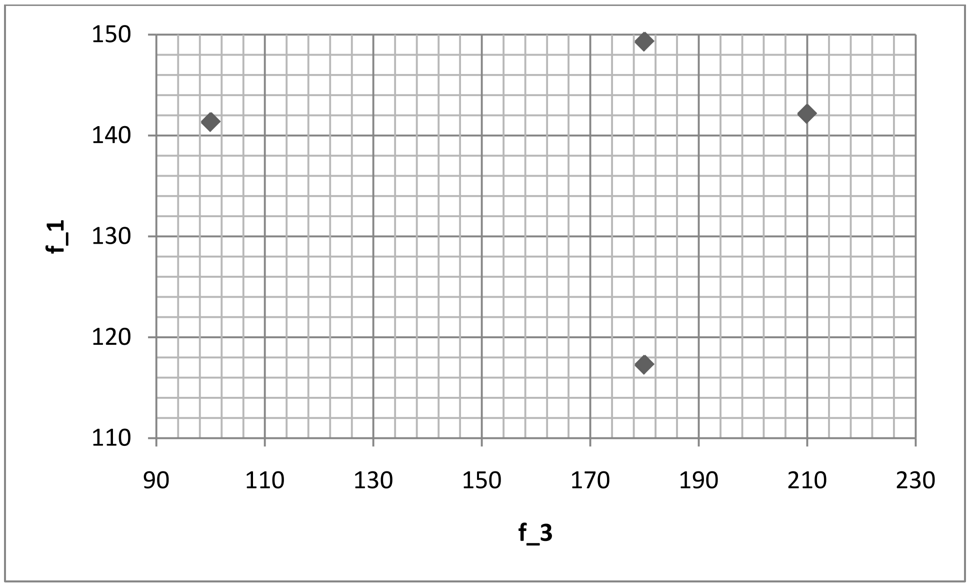


Fig.3.36 $f_3 - f_1$

Table 2.7.8 MAINTENANCE SCHEDULE

1st year	60	43	76	46	90	57	2	53	37	6
2nd year	32	67	83	44	80	52	15	65	86	11
3rd year	18	5	67	8	7	20	1	79	95	70
4th year	54	84	14	97	16	22	75	4	94	50
5th year	55	48	51	61	24	91	45	27	42	85

From above figures, we can see that NSGAII has been able to maintain a good spread of solutions (feasible and infeasible). However, when applied constraints, NSGAII can only get a few solutions feasible solutions.

Chapter 4 Conclusions

In the beginning of this thesis, an overview of Genetic Algorithms (GA) was presented. The main components, which play major roles in the algorithm, were briefly introduced. The aim of applying GA in this thesis is to minimize user delay of infrastructure maintenance plan. As a result, a thorough explain of Single Objective Genetic Algorithms (SOGA) was launched. It was separated into binary represented and integer represented parts to get the maintenance schedules. Thereafter, a very detailed explain of Multi-Objective Genetic Algorithms (MOGA) was presented. A fast and elitist MOGA (NSGAII) was used for the testing case.

The aim of this thesis is to create nearly ideal maintenance schedules by the use of genetic algorithms. If there is only one objective when making the schedules, which is called single objective, we applied both binary and integer represented scheme. Since the maintenance schedule should have no duplicated elements, the integer represented scheme led to better solutions with less duplicate bridge index and shorter traffic delay time (the objective). If two or more objectives exist, which is called multi-objective, NSGAII was applied to get the schedules. With the properties of a fast non-dominated sorting procedure, an elitist strategy, a parameter-less niching operator, NSGAII has found good spread out of the solutions. Further increasing of the population size got better spread of solutions.

REFERENCE

[1]. John Dalton, “Roulette wheel selection,” Newcastle Engineering Design Centre, Merz Court, Newcastle University, August 2010
[2]. Peyman Kouchakpour,” Population Variation in Canonical Tree-based Genetic Programming”, School of Electrical, Electronic and Computer Engineering, University of Western Australia,2008
[3]. Siddhartha K. Shakya,” Probabilistic model building Genetic Algorithm (PMBGA): A survey,” Technical Report, Computational Intelligence Group, School of computing, The Robert Gordon University, Aberdeen, Scotland, UK. pp.1, 2003
[4]. Assaf Zaritsky,” Introduction to Genetic Algorithms,” Ben-Gurion University, Israel
[5]. G. Winter,” Genetic algorithms in engineering and computer science,” 1995
[6].A. E. Eiben, “Introduction to Evolutionary computing”, Springer, pp.41, 2003
[7].M. Ehrgott. “ Multicriteria Optimization.” Springer, Berlin, second edition, 2005.ISBN 3-540-21398-8
[8].C. A. C. Coello, ”Evolutionary algorithms for solving multi-objective problems.” Springer, second edition, pp.7, 2007.
[9].K. Deb. “ Multi-Objective Optimization using Evolutionary Algorithms.” Wiley, pp.23-24, 2002
[10]. N. Srinivas, and K. Deb,” Multi-objective function optimization using non-dominated sorting genetic algorithms”, Evolutionary Computation, Vol. 2, pp. 221-248, 1995
[11].K. Deb,” Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II,” KanGAL Report No. 200001, Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur, Kanpur, PIN 208 016, India
[12]. E. Zitzler, and L. Thiele, “Comparison of multi objective evolutionary algorithms: Empirical results.” Evolutionary Computation, Vol.8. pp. 173-195, 2000
[13]. G. Rudolph, “Evolutionary search under partially ordered sets,” Technical Report No.CI-67/99, Dortmund: Department of Computer Science/LS11, University of Dortmund, Germany.
[14]. C. A. C. Coello. “ Evolutionary algorithms for solving multi-objective problems.” Springer, second edition, pp.91, 2007.
[15]. K. Deb, A. Pratap, and T. Meyarivan,” A fast and elitist multi-objective genetic algorithm: NSGA-II,” IEEE Transactions on Evolutionary Computation, 6(2):182-197, 2002
[16] Andrzej Jaszkiewicz ,”Genetic local search for multi-objective combinatorial optimization,” European Journal of Operational Research, Vol.137, Issue 1, pp. 50-71,2002