The Performance Analysis of AIS with Hypermutation and P-hypermutation in Solving Combinatorial Optimization, Comparing with Standard Bit Mutation EAs

Jinkun Li
Supervisor: Pietro Oliveto

A report submitted in fulfilment of the requirements for the degree of MSc in Advanced Computer Science
in the
Department of Computer Science

September 10, 2019

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amount to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Jinkun Li
Signature:
Date: 10th September 2019

Abstract

As a class of bionic heuristic algorithms, artificial immune systems (AIS) have been proven to have better performance than traditional evolutionary algorithms (EAs) in escaping local optima of some combinatorial optimization problems. It is because AISs add some complicated operators, such as aging and hypermutation. However, few experimental results in the literature clearly indicate whether this case. In this project, we encode to evaluate whether static hypermutations in (1+1) AIS outperform standard bit mutations in (1+1) EAs on a classical Combinatorial Optimization problem: job shop scheduling on 2 machines. The experimental results show that：

1. In the comparison between (1+1) EAs with standard bit mutation operators(SBM) and (1+1) AIS with hypermutation operators(HM), it is found that the hypermutation operator can better escape the local optimum when the number of iterations of the algorithm is sufficient. But the disadvantage is that the hypermutation operator is slower than standard bit mutation in the exploration phase.

2. The P-hypermutation operators(PHM) developed based on the static hypermutation operator can effectively improve the problem that the static hypermutation operator runs too slowly during the exploration phase.

。

Acknowledgements

Thanks to my supervisor Mr. Pietro Oliveto for guiding my dissertation project . He is a very warm-hearted and patient man.

Content

List of Figures

Chapter 1 Introduction

1.1 Background

Many combinatorial optimization problems are np-hard problems. They are easy to describe, but it is hard to calculate. For example, a classical combinatorial problem, called TSP, can be describe that a traveler needs to plan the shortest route to the city he plans to visit, and then return to the city he started from, and visit each city only once. This class of problems sounds easy, but with the increasing of the problem's size (a traveler wants to visit more cities), the optimization becomes more and more difficult (more routes can be selected), so that it cannot get the optimal solution on existing computers. Such problems are currently cannot be solved by suitable polynomial algorithms, but people can use heuristic algorithms to give a feasible solution at an acceptable cost [Bertsekas, 1997]. The two types of an algorithm to be considered in this paper are heuristic algorithms based on biological systems. A kind of algorithm is the artificial immune systems (AIS), which was proposed in the 1980s to solve complex mathematical problems [Farmer, 1986]. Another type of algorithm is evolutionary algorithms (EAs), which is also proven to have excellent performance in solving combinatorial optimization problems. As for optimization field, AIS is a subclass of the more general and well-known class of Evolutionary Algorithms (EAs). The main distinguishing feature compared to traditional EAs is the use in AIS of sophisticated operators such as hypermutations. It has been proven that AIS can efficiently optimize functions that are difficult for conventional EAs, [Corus, 2018]. For example, AIS can more effectively escape from local optimization. However, experimental results in the literature do not clearly explain whether this is the case.

1.2 Aims and Objectives

The project is an experimental, so the aims is to implement (1+1)AIS with hypermutation(HM) operator, (1+1)AIS with P-hypermutation(PHM) operator and (1+1)EAs with standard bit mutation(SBM) operator on a classical combinatorial optimization problem, job shop scheduling on 2 machines. Then, we evaluate which algorithm has better performance. Experimental results show that:

1. In the comparison between (1+1)EAs with SBM and (1+1)AIS with HM, it is found that the (1+1)AIS with HM can better escape the local optimum when the number of iterations of the algorithm is enough. But the price of this benefit is that the (1+1)AIS with HM is slower than (1+1)EAs with SBM in the exploration phase.

2. The (1+1)AIS with PHM developed based on the (1+1)AIS with HM can effectively improve the problem that the static hypermutation operator runs too slowly during the exploration phase.

1.3 Overview

There are six chapters in this dissertation:

Chapter 2. Literature Survey: This chapter describes the details about the project such as combinatorial optimization problems, artificial immune system and evolutionary algorithms.

Chapter 3. Requirements and analysis: This chapter describes the detail about the purpose of this project. Also, the requirements of the objectives will be discussed. Then, the method of testing and evaluation ethical, professional and legal issues and risk management associated with the project are discussed.

Chapter 4. Design: This chapter explains the design of three different algorithms and the fitness function of job shop scheduling on two machines. Then, experimental parameter design and experimental data processing methods will be discussed.

Chapter 5. Implementation and testing: This chapter introduces the implementation of experimental code and experimental test cases.

Chapter 6. Results and discussion: This chapter charts the results of experimental data. Then there are some discussion and compassion based on the experimental data.

Chapter 7. Conclusions: This chapter is a review of the work done and a summary of the experimental results. In addition, deficiencies in the work already completed were pointed out

Chapter 2 Literature Survey

2.1 Combinatorial Optimization Problems

The goal of combinatorial optimization problems is to find an optimal solution from the feasible solution set. let X= {x1, x2, ..., xn} to store the solution states, while F(X) is an objective function, which find the optimal solution from X so that F(X) can get a minimum value. Combinatorial optimization has many different problems, such as routing, scheduling. Classification and so on. Therefore, it is an essential branch of operations research [Papadimitriou, 1998]. A classical combinatorial optimization problem will be discussed in detail next.

A job shop scheduling problem is an NP-hard problem, so it is hard to calculate by polynomial methods [Graham, 1966.]. Its basic description is as follows: Suppose we have m tasks with different processing times (t1..., tn), and n machines with same speeds of processing. These tasks need to be proper scheduling on these machines and minimize the time spent on all assignments. This kind of problem exists in real life, for example: in the daily work of the workshop, it is often the case that several machines simultaneously handle a bunch of tasks. At this point, there is a problem that how to distribute these tasks to these machines reasonably makes the final completion time the shortest. When the number of jobs is small, workers can also schedule through simple calculations. However, as the number of jobs increases, the time complexity of JSP issues will increase rapidly.

Solving such problems can improve production efficiency, so many people are keen to explore how to solve this problem.Considering the difficulty of JSP and its practicality, researchers have created some heuristic algorithms to obtain an approximate solution with acceptable time [Hansen, 1990]. In recent years, biological systems have received more and more attention from researchers, and many heuristic algorithms based on biological systems have been proposed. For example, AIS are inspired by the natural biological immune system, while EAs is influenced by Darwinian evolution. These algorithms are proved to be effective in calculating JSP [Layeb, 2010].

2.2 Artificial Immune System (AIS)

2.2.1 Immune System Background

The immune system is a defense system against unknown viruses. In daily life, people may encounter some injuries, but the immune system can protect people's health. A primary function of IS is that the effective response to many virus intrusions with a limited resource. The process is described below[Janeway, 1996].

1. **Antigen recognition**: The immune system recognizes antigens and generates different plasma cells according to the characteristics of different antigens to produce antibodies.

2. **Select plasma cells**: if the antibody produced has a high affinity with the antigen, it will remain. Otherwise, it will be sieved off.

3. **Storage in memory cells**: Immunocyte differentiation and memory cells retain antibody information with high affinity.

4. **Control the production of antibodies**: plasma cells that produce high-affinity antibodies are promoted, and vice versa.

5. **Generate next-generation antibodies by cross mutation.**

2.2.2 Artificial Immune System Process

The AIS is constructed according to this defense mechanism of the human immune system [Kephart, 1994]. Table 2.1 shows their relationship.

Immune System	Artificial Immune System
Antigen	Optimization problem
Antibody	Feasible solution
Affinity	The quality of feasible solution
Cell activation	Immune Selection
Cell differentiation	Individual cloning
Affinity mature	Variation
Clonal inhibition	Excellent individual selection
Antibody refresh	Solution space update

Table 2.1 Relationship Between IS and AIS

Based on this correspondence, the flow chart of a basic artificial immune system structure is shown below. As the chart shows, operations on mutations, selection, clones, etc., are simplified as operators, while they will be described in the section of "Artificial immune system operators".

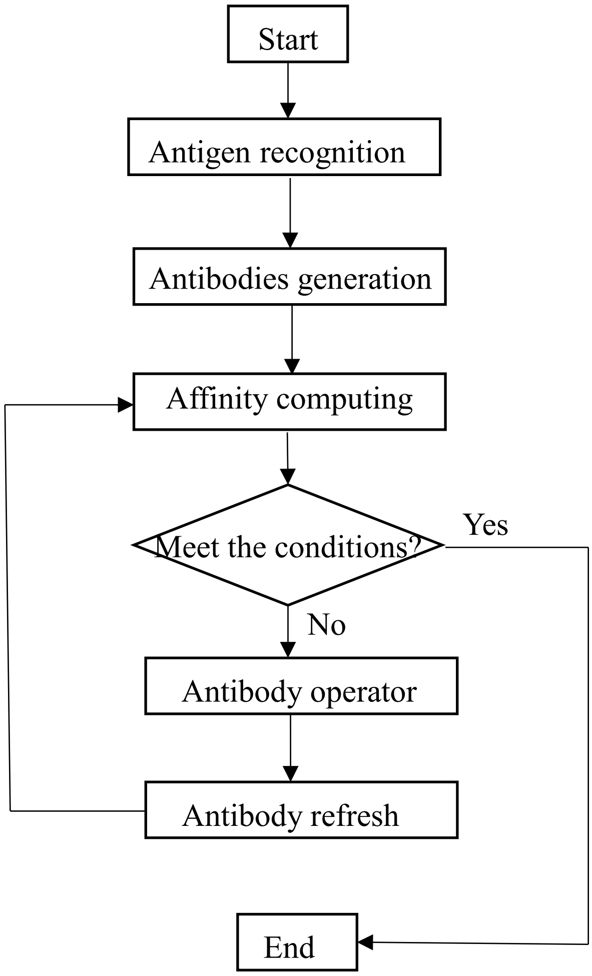

Figure 2.1 AIS Flowchart

2.2.3 Artificial Immune System Operators

Hypermutation Operator(HM)

This report focusses on the hypermutation operator. It is because this operator has been proven to have good performance for some benchmark functions in the literature [Basu, 2011]. As for the hypermutation operator, it acts on the B-cell differentiation, different types of hypermutations have been proposed as below [Cutello,2004]:

1. **Static**: The mutations are static, so each time the B cell mutates it's not going to exceed the static conditions.

2. **Proportional**: The mutations are proportional to the affinities. It means the lower the affinity, the higher the mutation rate, and vice versa.

3. **Inversely Proportional**: As opposed to proportional hypermutation, the higher the

affinity, the lower the mutation rate, and vice versa.

In the experiment of this project, this static hypermutation will be implemented and discussed, and he maximum flipped bit M is equal to n, which means that bit string can be completely flipped. Therefore , a simple example of static hypermutation is: Firstly, initialize a bitstring. Then, randomly flip a bit of a bit string if it has been evaluated for fitness. If bitstring after mutation is better, the hypermutation ends. Otherwise, the algorithm will continue to randomly mutate a bit in the part that has never mutated. Until you get a better bitstring or all the bits of the whole bitstring are flipped. The specific design of HM will be explained and displayed in detail in the design chapter

P-Hypermutation Operator(PHM)

PHM is developed based on HM. Some researchers found that HM would waste some iterations in the exploration phase of the algorithm, because HM would evaluate the fitness after each mutation[Corus, 2018]. PHM will calculate a probability after each mutation, and then the algorithm will decide whether to conduct fitness evaluation for this mutation according to this probability. This probability is not static, it is written as(1):

$$\boldsymbol{p}_i = \begin{cases} 1/e & (i = 1 \& i = n) \\ \gamma/i & (1 < i \leq n/2) \\ \gamma/(n-i) & (n/2 < i < n) \end{cases} \tag{1}$$

The specific design of PHM will be explained and displayed in detail in the design chapter.

2.2.4 The Application of AIS

Many literatures have conducted some benchmark problem tests on AIS to analyze the expected running of AIS when solving some toy problems. For example, through rigorous theoretical analysis, some people proposed the expected running time of AIS with static hypermutation when solving OneMax, LeadingOnes, Trap, and other problems, which are shown in the figure[Thomas, 2011].

Function	$(1+1)AIS^{HM}$
$ONEMAX$	$\theta(n^2 logn)$
$LEADINGONES$	$\theta(n^3)$
$TRAP$	$\theta(n^2 logn)$

Table 2.2 The Expected Runtime of $(1+1)AIS^{HM}$

In addition to testing some toy problems, many researchers have applied AIS to more practical problems, especially some combinatorial optimization problems. AIS has been widely used in solving combinatorial optimization problems and achieve good performance. For example, Shih-Wei Lin and KUO-ching Ying [Lin, 2013, pp.383-389] implemented a revised AIS to find a suitable solution for the blocking flow shop scheduling while the AIS get better performance than most algorithms. Another example is about the performance of AIS in solving TSP. TSP is widely used to verify the performance of many heuristic algorithms as a benchmark problem. As for AIS, some researchers use TSP to test the performance of clonal selection algorithm (a type of AIS), and the result is satisfactory [De Castro, 2002, pp. 239-251]. All these examples prove the feasibility of our project.

2.3 Evolutionary Algorithms (EAs)

2.3.1 Evolutionary Algorithms Background

Like artificial immune systems, evolutionary algorithms are also biological heuristic optimization algorithms. It is inspired by the theory of natural evolution, with operators such as reproduction and selection [Back, 1996]. And this algorithm does not require an adaptive environment, so it has a good approximate solution on most optimization problems. Its necessary steps are 1. Generate an initial population. 2. Comment on the individual's adaptability to the environment. 3. Select the most adaptable individuals for breeding, and there will be crossover and mutation during the breeding process to nurture new individuals. 4. Evaluate unique individuals. 5 Replace the worst individual with a new individual. In this process, the initial population is the initial solution space. Then, by performing the operator operation of the evolutionary algorithm on the individuals in the initial solution space, a suitable approximate solution is finally generated.

2.3.2 Standard Bit Mutation Operators(SBM)

Standard bit mutation(SBM) is a classical EAs operator, is a common mutation operator of evolutionary algorithm. It is simply described as follows: first, initialize a bit string of length n. Then flip each bit of the bit string with a probability of 1/n. From the perspective of mathematical expectation, the expected number of mutations of each bit string mutation is 1. The specific design of SBM will be explained and displayed in detail in the design chapter

2.3.3 The Application of EAs

In many literatures, researchers also analyzed expected running time of EAs with SBM.

Based on the tests of some toy problem, the test results are shown in the table, which contains the analysis results of AIS with static hypermutation for convenience of comparison. As can be seen from the table, the expected running time difference between EAs with and AIS with HM when dealing with these problems is shown[Stefan, 2002]. In the OneMax problem and LeadingOnes problem, the expected running time of EAs with SBM is less than that of AIS with HM.AIS with HM is better for Trap problems.

Function	$(1+1)AIS^{HM}$	$(1+1)EAs^{SBM}$
$ONEMAX$	$\theta(n^2 logn)$	$\theta(n\ logn)$
$LEADINGONES$	$\theta(n^3)$	$\theta(n^2)$
$TRAP$	$\theta(n^2 logn)$	$\theta(n)$

Table 2.3 The Expected Runtime of $(1+1)AIS^{HM}$ and $(1+1)EAs^{SBM}$

EAs also has excellent performance in solving combinatorial optimization problems. As early as in the 1980s, there was research on how to solve TSP (a kind of combinatorial optimization problems) with EAs [Grefenstette, 1985]. Nowadays, various improved EAs are used to solve combinatorial optimization problems. For example, a hybrid genetic algorithm (GA, a type of EAs) was proposed. It co-optimizes GA and ant colony optimization algorithm so that GA has better search accuracy and faster convergence [Deng, 2017]. From these two examples, after such a long period of development, EAs is still active in solving combinatorial optimization problems, indicating that it is still an algorithm worth studying.

2.4 Chapter Conclusion

In this chapter, it is a detailed description of the concepts of combinatorial optimization, AIS and EAs. Meanwhile, this chapter highlights some of aspects which have been emphasized based on the requirement of project aims.

Chapter 3: Requirements and analysis

3.1 Aims and Objectives

The project aims to analyze the performance of (1+1)AIS with static hypermutation operators(HM) and P-hypermutation operators(PHM) comparing with (1+1)EAs with standard bit mutation operators(PHM). While job shop scheduling (JSP) on two machines is used to detect the performance of these algorithms in solving combinatorial optimization problems and highlighting the advantage of (1+1)AIS in escaping local optimal.

Based on a brief description of the previous paragraph, the project will implement the three different algorithm functions and benchmark functions at first. Then, collect the data by testing the benchmark problem by the implemented algorithm code. Finally, verify the conclusions in the literature by analyzing the data obtained.

3.2 Requirements

Requirements are the functions that must be provided in the project or the conditions to be followed. In actual use, the requirements can be subdivided into functional and non-functional requirements.

3.2.1 Functional Requirements

Functional requirements specify the functions that developers must implement in their projects, and users use these features to accomplish tasks. In this project, users can choose different algorithm models to test benchmark problems of different scales. In addition, users can choose different maximum iterations, number of tests, and so on.

The following functional requirements relate to user operations:

• User can choose benchmark problem files from their local disk by changing the read path of the file.
•User can change the path to save the generated data file.
•User can change the filename of the generated data file.
•User can choose different max iterations.
•User can choose the number of instances.
The following functional requirements relate to data analysis:

•The output includes average solution of benchmark problem in every instance.
•The output includes best solution of benchmark problem in every instance.
•The output includes worst solution of benchmark problem in every instance.
•The output includes medium solution of benchmark problem in every instance.

3.2.2 Non-functional Requirements

Non-functional requirements refer to the characteristics that projects must have in addition to functional requirements in order to meet the user's business needs. The requirements of this project are as follows:

•**Scalability:** This experiment needs to test three different algorithm models on benchmark problems of different sizes, so the program needs good scalability. Programs can adjust some of the parameters in the code to achieve different combinations. For example, output the experimental results of two algorithms, or output the experimental results of three algorithms.

•**Robustness:** There are many detection codes in the important steps of the program that display the current state of the program. Through the displayed information, the errors in the program can be easily modified to achieve code robustness. These detection codes are commented out in the code of the submitted version.

•**Response time:** Response time is primarily related to two factors, one of factor is the performance of the hardware. On hardware which have different performance, the response time of the same program is different. And even on the same hardware, the response time is different because the state of the hardware is not the same. In this case, in order to reduce the response time of the application, the program need run in the same hardware, and shut down other unrelated applications while the program is running. Another factor that affects the response time is the optimization of the algorithm. We should optimize the program to reduce bugs and redundant loops.

•**Replicability:** The results of this experiment are not a special case but can be repeated on different hardware. This is because the experimental code is based on the Python3 standard library, including the math library and the random library. These libraries are universal on different hardware

•**Code reusability:** To ensure that the code for this project can be reused in future work, important parts of the project are written as separate modules. These modules can be reused in other code to avoid unnecessary duplication of effort. For example, the file read and write function, the benchmark problem function, and three different algorithm models, they can all be reused with simple modifications and testing. In addition, there are numerous comments in the code that help explain what the code does

3.2.3 Analysis of the Objectives

The overall objectives of the project have been described, but the steps before the main goal achieved should also be analyzed. Therefore, the analysis of the steps of the project is as follows:

1. Build a benchmark test, one max function. The purpose of building this benchmark problem is to verify the correctness of the three algorithm models built next. The one max problem is a very common benchmark test for testing the performance of heuristic algorithms, and the test based on OneMax problem can reflect the single peak climbing ability of three algorithms. If the algorithm cannot obtain the best fitness in the OneMax benchmark, the algorithm is wrong. In this way, the correctness of the algorithm can be verified

2. Construct three algorithm models. Of note is the implementation of random numbers in these algorithm models. All random numbers in this project are based on the uniform function in the python3 random library, which uses Mersenne Twister as a random number generator. The literature shows that different pseudo-random number generators have little impact on the performance of heuristic random search algorithms[Cantú-Paz, 2002]. Therefore, it is feasible to use this random number to implement the algorithm.

3. Build a benchmark test problem, the function of JSP on two machines.

4. Perform OneMAX benchmark tests on the three algorithm models built to ensure the correctness of the algorithm model. In this test, you should consider the size and number of tests of the OneMAX problem, and the appropriate parameters can get good test results.

5. Perform JSP on two machines benchmark tests on the three algorithm models built and save the result data in an excel table. This step should consider the three parameters max iteration, test time, instance number. The setting of the parameters should be considered in the design section.

6. Make appropriate charts based on the resulting data to compare and highlight the performance of the three algorithms

3.2.4 Evaluation and Testing

In order to ensure that the build of the program meets the functional and non-functional requirements of the project, it is important to evaluate and test in a timely manner during the construction of the experiment. Therefore, the following test methods will be used.

•**White box testing:** The white box testing is mainly used in the unit test phase, mainly for the code level test, for the internal logic structure of the program, the test means are: statement coverage, decision coverage, condition coverage, path coverage, conditional combination coverage. The white box test is also called the structure test or the logic drive test. It is based on the internal structure test program of the program. It is tested to check whether the internal motion of the product is normally performed according to the specifications of the design specification. It is possible to check whether each path in the program can be pressed. The booking is required to work correctly. This method is to treat the test object as an open box. The tester designs or selects the test case according to the information about the internal logic structure of the program, tests all the logical paths of the program, and checks the state of the program at different points to determine the actual Whether the status is consistent with the expected status.

In this project, the main purpose is to test whether each function runs normally. For file read/write systems, data consistency needs to be tested. For the algorithm model, the correctness of each step in the model needs to be tested. For the benchmark problem module, we need to test whether their fitness calculation is correct.

•**Black box testing**: The black box test does not consider the internal structure and logical structure of the program and is mainly used to test whether the function of the system meets the requirements specification. There will generally be one input value, one input value, and an expected value. The black box test is also called the function test. It is tested to check whether each function can be used normally. In the test, the program is regarded as a black box that cannot be opened. It is tested at the program interface without considering the internal structure and internal characteristics of the program. It only checks whether the program function is normally used according to the requirements specification. Whether the program can properly receive input data to produce correct output information. The black box test focuses on the external structure of the program, regardless of the internal logic structure, and is mainly tested for software interfaces and software functions.

In this project, the most important part is the three algorithm models, so after building the algorithm model, we will use the ONEMAX benchmark problem of 100szie to test the algorithm model. The ONEMAX model can investigate the single peak climbing ability of the algorithm. If the algorithm model is correct, then the three algorithm models can reach the peak, that is, all the bits of the bitstring are 1.

3.3 Ethical, Professional and Legal Issues

Personnel in computer science should be fully aware that their research must comply with the law and be subject to ethical constraints. To achieve this goal, researchers should adhere to the following code of conduct [Mason, 2017]:

1. Researchers must ensure the validity, integrity, authenticity and safety of their research data.
2. Researchers should justify their research data before draw conclusions.
3. Researchers must respect intellectual property rights and not copy or steal other people's research results.
4. Researchers should ensure that their research topics are by human ethics.
5. Researchers must protect the privacy and legal rights of others.

As for the project, there is no ethical problem because of the followings:

1. The project mainly discusses the efficiency of the artificial immune algorithm and evolutionary algorithms in solving the combinatorial optimization problem. While the thesis was proposed by my supervisor.
2. software used in the research project (pyhon3 and VS code) are open, so there have no copyright issues.
3. The initial data of the experiment will be randomly generated by Python codes or getting from open Data sets library (Follow the rules of reference), so there are no issues with the data source.
4. The project will not involve other people's privacy issues.

3.4 Risk Management Plan

A risk is any threat to the achievement of a project goal. In other words, risk management is good to achieve project objectives, such as improving work efficiency, project stability and security. As for this project, there are some cases of risk which may bring to different impact of the project. The table shows the analysis of these risks:

Rank	Risk	Likelihood	Impact	Exposure	Action
1	Failure to implement algorithm code	2	4	4	Take enough time to learn related technologies

2	Code running too slowly	2	3	3	Upgrade hardware or use university's cloud computing resources
3	code missing	1	3	4	Use GitHub to save and manage code
4	Failure to meet all of project aims	2	4	3	Analyze in detail before starting the implementation
5	Not finishing the work as planned	3	3	2	Make a time management plan

Table 3.1 Risk Management Plan

3.5 Chapter Conclusion

The first section of this chapter introduces the main objectives of the project. Then, in chapter 3.2 and 3.3, the requirements and analysis of the objectives will be discussed. Finally, it will state the ethical issues and risk management plan.

Chapter 4: Design

4.1 Input Representation

In order to ensure the generality of the algorithm, it is very important to make sure how to input the benchmark problem into the algorithm models. The bit string $\{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}}$is a good input representation type. The input of OneMax benchmark problem seems simple because it simply inputs a list which includes$\{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}}$into the algorithm model. For example, in a OneMax problem which is 5 size, if the initial list is **{0, 1, 0, 1, 1}**, the fitness is **3**.Then, mutation happened, and the new list after mutation operation is **{1, 1, 0, 1, 1}**, the fitness is **4**.This string of data consisting of 0,1 is called bit string.

The Bit string method also applies to job shop scheduling on two machines. Suppose that there are five jobs: **{j0, j1, j2, j3, J4}**, and two machines: **{m0, m1}**. The time required for the machine to process these five jobs is **{t0, t1, t2, t3, t4}**. Now **0** is **m0, 1** is **m1.** While a bit string **{0, 1, 0, 1, 1}** means m0 is responsible for **{j0, j2}**, and m1 is responsible for **{j1, j3, j4}**. So, their total working time: **T = Max (t0+t2, t1+t3+t4)**, which is also called fitness in EAs and AIS. From the above discussion, the two problems of OneMAX and JSP on two machines can be abstracted as bit strings composed of 0 and 1. The difference between them lies in the calculation of fitness. The design of their fitness functions will be discussed later.

4.2 Fitness Function Design

As for evolutionary algorithms, artificial immune algorithms and other biological heuristic random search algorithms, the evaluation of a solution depends not on the form of the solution, but on the fitness of the solution. The algorithm evaluates the solution based on its fitness. It is of great significance in the evolution process. By mapping the objective function of the optimization problem with the individual's fitness, it can be in the group.

The fitness of OneMax benchmark problem is the number of 1 in the current bit string. It can be written as:

$$\text{OneMax}(x) \coloneqq \sum_{i=1}^{n} x_i \qquad (2)$$

Based on the formula, the pseudo-code of fitness function of OneMax bench mark problem is shown as figure 4.1. The pseudo-code illustrates how the fitness of the OneMax problem is calculated. First, get the current bit string and its length from outside the fitness function. Then add each bit of the bit string through a loop.

Fitness function of OneMax benchmark problem

Get the current bit string from the algorithm module x: = $(x_1,\ x_2,\ \dots,\ x_i)$
Get the length of x, len: = length(x)
i: = 0
while i < len **do**
    fitness: = fitness + x_i
    i: = i + 1
end while

Figure 4.1 The Pseudo-code for Fitness Function of OneMax

Comparing with the fitness of OneMax, the fitness of job shop scheduling (JSP) on two machines is more complex. Suppose that there are n jobs: $j_0, j_1,\ \ldots,\ j_n$. And their processing time: $t_0, t_1,\ \ldots,\ t_n$. Now assign these tasks to two equally efficient machines, each responsible for a different task. In bit string $x\ \in \{0,1\}^n$, we use $x_i = 0$ to represent a machine m0, and $x_i = 0$ to represent the other machine, so the fitness function can be written as follows:

$$\mathrm{JSP}(x) \coloneqq \max\left(\textstyle\sum_{i=1}^{n} t_i x_i, \sum_{i=1}^{n} t_i (1 - x_i) \right) \qquad (3)$$

Therefore, the pseudo-code of the fitness of OneMax benchmark problem is shown as figure 4.2.

Fitness function of JSP on two machines benchmark problem

Get the current bit string from the algorithm module x: = $(x_1,\ x_2,\ \dots,\ x_i)$
Get processing time data from outside the function t: = $(t_1,\ t_2,\ \dots,\ t_i)$
Get the length of x, len: = length(x)
i: = 0
Total working time of machine 0, T_0: = 0
Total working time of machine 1, T_1: = 0
while i < len **do**
    if x_i == 1
        $T_1 := T_1 + t_i$
    end if
    if x_i == 0
        $T_0 := T_0 + t_i$
    end if
    i: = i + 1
end while
fitness: = max (T_0, T_1)

Figure 4.2 The Pseudo-code for Fitness Function of JSP on Two Machines

4.3 Algorithm Function Design

Three algorithms need to be designed and implemented in the project: (1+1) AIS with static hypermutation operators, (1+1) AIS with P-hype FCM operators, and (1+1) EAs with standard bit mutation operators. This section shows the simple pseudo-code for these three algorithms and details how to design the special features of the three algorithms.

4.3.1 (1+1) EAs with standard bit mutation (SBM)

For the special features of this algorithm, SBM operator will first calculate a probability according to the length of bit string, and then every bit of bit string will mutate as a probability. Mathematically, bitstring only flips one bit at a time. Based on this feature, simple pseudo-code of EAs with standard bit mutation (SBM) has been revealed in figure 4.3.

Algorithm (1+1) EAs with standard bit operator

```
Initialize bit streaming x: = (x1, x2, …, xn);
Evaluate f(x);
while termination condition not satisfied do
    y: = x;
    Flipping y each bit with probability 1/n;
        Evaluate f(y);
    end while
    if f(y) better than f(x) then
        x: = y;
    end if
end while
```

Figure 4.3 The Pseudo-Code for (1+1) EAs with Standard Bit Mutation

4.3.2 (1+1) AIS with hypermutation (HM)

The special feature of (1+1) AIS with hypermutation (HM) has two points. Firstly, comparing with standard bit mutation, Hypermutation is characterized by a higher mutation rate. The SBM operator flips one bit at a time, mathematically.HM operator has the opportunity to reverse the whole bit string.

In addition, first construct mutation (FCM) strategy has been applied in (1+1) AIS with hypermutation operators. The implication of this mutation is that after each mutation. If the solution improves, the solution is immediately updated, and the old solution discarded. In other words, for each mutation, the algorithm will calculate the fitness after mutation, and then decide not to update the solution according to the fitness. This strategy has been shown to be very effective in hypermutation operators, so I use this strategy to improve the performance of the algorithm[D. Corus, 2017].

Based on the two points in the last paragraph, the pseudo-code of (1+1) AIS with hypermutation (HM) is shown as figure 4.4.

Algorithm (1+1) AIS with hypermutation operator

```
Initialize bit streaming x: = (x1, x2, …, xn);
Evaluate f(x);
while termination condition not satisfied do
    y: = x;
    F: = {1,2, …, n};
        while F ≠ ∅ and f(y) poor than f(x) do
            i: = Select randomly from F;
            Flipping y in bit i; delete i from F;
            Evaluate f(y);
        end while
        if f(y) better than f(x) then
            x: = y;
        end if
end while
```

Figure 4.4 The Pseudo-Code for (1+1) AIS with Hypermutation

4.3.3 (1+1) AIS with P-hypermutation (PHM)

PHM operator is developed on the basis of HM operator, so it also has two special features of high mutation rate and FCM strategy. However, the biggest difference between PHM operator and HM operator is that HM operator will calculate and evaluate the fitness of bitstring after every mutation, while PHM evaluates the utility with a certain probability. In other words, not all mutation operations are evaluated for fitness. The probability density function of evaluation has been introduced in detail in the literature review, and its formula is as follows:

$$\boldsymbol{p}_i = \begin{cases} 1/e & (i = 1 \& i = n) \\ \gamma/i & (1 < i \leq n/2) \\ \gamma/(n-i) & (n/2 < i < n) \end{cases} \qquad (4)$$

Based on the probability density function, the (1+1) AIS with p-hypermutation pseudo-code is shown in the figure 4.5.

Algorithm (1+1)AIS with P-hypermutation operator

```
Initialize bitstring x: = (x_1, x_2, …, x_n)
Evaluate f(x);
while termination condition not satisfied do
    y: =x;
    F: = {1, 2, …, n};
        while F ≠ ∅ and f(y) poor than f(x) do
            i:= Select randomly from F;
            Flipping y in bit i; delete I from F;
            Calculate current probability of evaluation
            Evaluate f(y) by calculated probability
        end while
        if f(y) better than f(x) then
            x: = y;
        end if
end while
```

Figure 4.5 The Pseudo-Code for (1+1) AIS with P-Hypermutation

4.4 Research Methodology

4.4.1 Programming Tools

During the project, python3 and Visual Studio code are our research tools. Python is a compelling language that has been widely used in algorithms implementation and data analysis in decade years [Chou, 2002. p.2]. And thanks to the open source features of Python, it has several algorithm libraries available, which will help the fast start and progress of research, such as DEAP [Fortin, 2012, pp.2171-2175]. And there are many precedents for using python for artificial immune algorithms, evolutionary algorithms to implement and assign. For example, Terri Oda has implemented AIS to build a system foe Junk Email Detection by Python in 2004 [Oda, 2005, pp. 276-289]. Therefore, python is a good choice. VS code is a cross-platform editor released, and It is rich in features and python support.

4.4.2 Benchmark Testing Methods

The project is to implement an algorithmic program in the Python3 language, and then to input an initial data set into the program to evaluate the performance of the algorithmic program. Therefore, the following factors need attention.

Dataset: The dataset will be considered from the following two aspects: 1. Randomly generated by Python code and controlled by the random seed to ensure the reasonableness of the dataset. 2. Get the right dataset from an existing open database site. In this experiment of JSP on two machines problems, the experimental data comes from the website http://mistic.heig-vd.ch. This website provides many data sets for combinatorial optimization problems, and these problem sets are also used by many other researchers.

The termination condition: The termination condition of the algorithm is essential to determine when the algorithm runs. There are two considerations: 1. The setup algorithm terminates in the X iteration. 2. Set the algorithm to end when the X iteration results are not improved. These two termination conditions will be used in the implementation phase and depending on the quality of the code. This experiment uses the former method

Results evaluation: The analysis of the results should be considered in two aspects: the time consumption of the algorithm under the same termination conditions; the result qualities of the algorithm results under the same number of iterations. Also, performance evaluation should be presented in the form of charts, such as histograms, line charts, etc. After evaluating the performance based on the two aspects, the reasons for such performance differences should also be analyzed. For example, we can test the impact of data sets of different max iteration on the results.

4.5 Experiment Design

From section 4.1 to 4.3, it is detailed discussed the design of the program, which section 4.4 will introduce how to design the experiment, including the design of parameter and the design of data collection.

4.5.1 Parameters and Variables

The purpose of the experiment is to test the performance of the three biometric heuristic search algorithms on the JSP on two machines problem. Therefore, in order to eliminate unnecessary interference and obtain the most authentic data as possible, i Some parameters need to be manually adjusted during the experiment.

.

MaxIteration: It can be seen from the pseudo-code in this chapter that the termination condition of the algorithm is required. For this experiment, the termination condition of the algorithm is the maximum iteration number. When the number of iterations of the algorithm is equal to the set number of iterations, the problem test of the current instance and number ends. The program then records the fitness of the solution when the algorithm stops. The purpose of the experiment is to highlight the ability of the algorithm to escape the local optimal, so a series of maximum iteration times should be set to observe the ability of the algorithm to escape the local optimal under different maximum iteration times. Four different iterations were set in this experiment: 300, 500, 1000, 2000.

InstanceNumber: In order to make the experimental data as accurate as possible, the algorithm should be tested in as many instances as possible. This experiment selects 100 JSP benchmark problem instances from e. Taillard's "Benchmarks for basic scheduling problems".

TestTime: Similarly, for the accuracy of the data, each problem instance was tested 100 times, and the best fitness, worst fitness, intermediate fitness and average fitness of each test instance were obtained from the 100 tests.

Problem Size: Two points should be taken into account when selecting the size of the benchmark problem. First, the base problem should not be too small. Too small size is not conducive to observing the performance difference of the three algorithms. Secondly, size should not be too large. Too large size will slow down the progress of the experiment, and a large amount of time is wasted in the collection of experimental data. I've selected the 100size problem set.

The Settings for these parameters are summarized as table 4.1

Max Iteration	300, 500, 1000, 2000
Problem Size	100
Instance Number	100
Run Time	100

Table 4.1 Adjustable Parameters

4.5.2 Data Collection Design

The purpose of the experiment is to compare the performance of the three algorithms to highlight their ability to escape the local optimal. Therefore, the data collected should reflect their performance at different maximum iterations. The performance of solutions is measured by calculating and comparing their fitness, so the performance of the algorithm can be measured by comparing the fitness values at different maximum iterations. This experiment, there are a lot of repetitive experiments for each benchmark problem instances, therefore, we can collect them in 100 times repeated the experiment of the fitness of each time, and then through the 100 crossovers. Simulation they reached the number of optimizations, the average fitness, the value of fitness and worst fitness and the best fitness. Then put these data into the form of output. The table 4.2 header of the table looks like this.

Instance	Max iteration	Best fitness	Average fitness	Medium fitness	Worst fitness

Table 4.2 The Header of Experimental Result Data Sheet

4.6 Read and Write Function Design

The data read function considers how to read external data. The external data used in this experiment come from 100 JSP problems in the website http://mistic.heig-vd.ch.Each of these 100 instances contains the running time to process 100 jobs, so the data is stored in the file as TXT. There are 100 lines in the txt file, and each line contains 100 variables. So, the program reads the file function can read the TXT file, and save the data in the file in the data structure.

The data write function considers how to output the resulting data in a reasonable form. The method adopted in this experiment is to output the data obtained from the experiment to the XLS file in the form of a table. This method can save all experimental data and facilitate the graphical display of these data.

4.7 Chapter Conclusion

The first section of this chapter introduces the input representation of benchmark. Then, there are some pseudo-code and explanation of EAs with SBM, AIS with HM, and AIS with PHM in chapter 4.2 and 4.3. After that chapter 4.4 and 4.5 introduce the design and method of experiment. Finally, the design of file reading and writing function is explained

Chapter 5: Implementation and testing

5.1 Algorithm Implementation

The implementation of algorithm code is based on the pseudo-code of design chapter. In order to improve the efficiency of the code, we improve the pseudo-code.

5.1.1 Standard Bit Mutation Operators

Two parts of the code implementation are special comparing with simple pseudo code. The first is the implementation of the randomness of the mutation operator. For SBM operators, it is important to evaluate the probability of mutations in each bit mutation. Uniform function of the random module is used in this part of the code implementation. The function guarantees the fairness of probability calculations. The second is to improve the efficiency of the algorithm and reduce unnecessary loops. For the JSP on two machines problem, it is not necessary to traverse the entire bitstring every time in fitness calculated. Simply checking the seat of the mutation and then modifying it before the mutation is sufficient. Therefore, this algorithm function will return a list of the seat where the mutation occurred during this mutation, and it will be provided to the fitness function. The specific Python3 code implementation is shown below.

```
def SBM_operator_JSP(list_bitstring):
    list_bitstring_cpy = list_bitstring.copy()
    global sum_0, sum_1, sum_0_cpy, sum_1_cpy, pro, jobs_num
    sum_0_cpy = sum_0
    sum_1_cpy = sum_1
    list_mutation = []
    for j in range(jobs_num):
        buf = random.uniform(0,1)
        if buf <= pro:
            list_mutation.append(j)
            if list_bitstring[j] == 1:
                list_bitstring_cpy[j] = 0
            if list_bitstring[j] == 0:
                list_bitstring_cpy[j] = 1
    return list_bitstring_cpy, list_mutation
```

5.1.2 Hypermutation Operators

As for HM operators, the difference between implementing code and pseudo-code is the mutation bit selection strategy. In the simple pseudocode, a list is created firstly, and then each time of hypermutation randomly selects a variable from this list, while this variable will be removed from the list after mutation. In the actual code implementation, considering that the modification of the list will take a lot of time, the strategy of mutation bit selection is changed. Generate a list firstly. Variables of the list are not arranged in order from smallest to largest, but will be shuffled by the shuffle function built in Python. The program then mutates from the first variable in the list until fitness improves or the entire bitstring is flipped. The specific Python3 code implementation is shown below.

```
def Hyper_operator_JSP(list_bitstring, list_flip, fitness_old, list_JSP, iteration):
    global sum_0, sum_1, sum_0_cpy, sum_1_cpy, cn
    sum_0_cpy = sum_0
    sum_1_cpy = sum_1
    length = len(list_bitstring)
    list_bitstring_cpy = list_bitstring.copy()
    list_flip_cpy = random.sample(list_flip, length)
    seat = buf = pro = iteration_HM = 0
    fitness_new = fitness_old
    list_mutation = []
    while(seat < cn and fitness_new >= fitness_old   and iteration + iteration_HM
< maxiteration):
        flipindex = list_flip_cpy[seat]
        list_mutation.clear()
        list_mutation.append(flipindex)
        buf = list_bitstring_cpy[flipindex]
        if   buf == 1:
            list_bitstring_cpy[flipindex] = 0
        if   buf == 0:
            list_bitstring_cpy[flipindex] = 1
        fitness_new = fitness_JSP(list_bitstring, list_JSP, list_mutation)
        iteration_HM = iteration_HM + 1
        seat = seat + 1
    return list_bitstring_cpy, fitness_new, iteration_HM
```

5.1.3 P-hypermutation Operators

PHM operator is improved on HM operator, so PHM operator function is modified on the basis of HM operator function when implementing PH. Compared with HM operator function, two points are modified. Firstly, before the fitness function evaluation, the probability of fitness evaluation will be calculated according to the probability density function proposed in the design section, while $\lambda = 1 / e$. Secondly, the fitness function is not evaluated every time, but mutations occur every time. Therefore, the list of mutation seats recorded will only be reset after the fitness evaluation occurring, while the reset operation in the HM function occurs before each mutation. The specific Python3 code implementation is shown below.

```
def P_hyper_operator_JSP(list_bitstring, list_flip, fitness_old, list_JSP, iteration):
    global sum_0, sum_1, sum_0_cpy, sum_1_cpy, cn
    sum_0_cpy = sum_0
    sum_1_cpy = sum_1
    length = len(list_bitstring)
    list_bitstring_cpy = list_bitstring.copy()
    list_flip_cpy = random.sample(list_flip, length)
    seat = buf = pro = iteration_HM = 0
    fitness_new = fitness_old
    list_mutation = []
    while(seat < cn and fitness_new >= fitness_old   and iteration + iteration_HM
< maxiteration):
        flipindex = list_flip_cpy[seat]
        list_mutation.append(flipindex)
        buf = list_bitstring_cpy[flipindex]
        if buf == 1:
            list_bitstring_cpy[flipindex] = 0
        if buf == 0:
            list_bitstring_cpy[flipindex] = 1
        if seat   == 0 or seat == 99:
            p = 1 / math.e
        if seat > 0 and seat <= 49:
            p = 1 / (math.e * (seat + 1))
        if seat > 49 and seat < 99:
            p = 1 / (math.e * (99 - seat))
        p_buf = random.uniform(0,1)
        if p_buf <= p :
            fitness_new = fitness_JSP(list_bitstring, list_JSP, list_mutation)
            iteration_HM = iteration_HM + 1
            list_mutation.clear()
        seat = seat + 1
    return list_bitstring_cpy, fitness_new, iteration_HM
```

5.2 Fitness Function Implementation

5.2.1 JSP on Two Machines Fitness Function Implementation

As for JSP on two machines fitness function, besides the external input of bitstring, JSP problem information, as well as the mutation seats information should be obtained. As stated in the algorithm implementation section, the JSP two machines fitness function will modify fitness according to the list of mutation locations. The specific Python3 code implementation is shown below.

```
def fitness_JSP(list_bitstring, list_JSP, list_mutation):
    global machines_0, machines_1, sum_0, sum_1, sum_0_cpy, sum_1_cpy
    length_mutation = len(list_mutation)
    if length_mutation > 0:
        for i in range(length_mutation):
            buff = list_mutation[i]
            if list_bitstring[buff] == 0:
                sum_0_cpy = sum_0_cpy - int(list_JSP[machines_0][buff])
                sum_1_cpy = sum_1_cpy + int(list_JSP[machines_1][buff])
            if list_bitstring[buff] == 1:
                sum_0_cpy = sum_0_cpy + int(list_JSP[machines_0][buff])
                sum_1_cpy = sum_1_cpy - int(list_JSP[machines_1][buff])
    fitness = max(sum_0_cpy, sum_1_cpy)
    return fitness
```

5.2.2 OneMax Fitness Function

OneMax fitness function is to calculate the number of 1 in bitstring. The code logic is the same as the pseudo-code logic, while the specific Python3 code implementation is shown below.

```
def fitness_OneMax(list_bitstring):
    length_bitstring = len(list_bitstring)
    for i in range (length_bitstring):
            fitness = fitness + list_bitstring(i)
    return fitness
```

5.3 Reading and Writing Function Implementation

5.3.1 Reading Function Implementation

The reading function reads the JSP problem data set from a text file. The data set is manipulated by this function and stored in a two-dimensional list whose rows represent test instances. The specific Python3 code implementation is shown below.

```
def readFile_JSP(fpname):
        f = open(fpname)
        list_JSP = []
        for lines in f.readlines():
                temp1 = lines.strip("\n")
                temp2 = temp1.split()
                list_JSP.append(temp2)
        f.close()

        return list_JSP
```

5.3.2 Reading Function Implementation

The writing function will output the experimental data to an XLS table .In python3, it is necessary to import an xlwt module to create and operate a table. Since the data types required by the experiment are fixed, including instance number, the maximum number of iterations, the best fitness, and the average fitness, the worst fitness and the intermediate fitness of the three algorithms. The specific Python3 code implementation is shown below.

```
import xlwt
f = xlwt.Workbook()
sheet1 = f.add_sheet(u'sheet1',cell_overwrite_ok=True)
for i in range(len(instance)):
    sheet1.write(i,0,instance[i])
for i in range(len(max_iteration)):
    sheet1.write(i,1,max_iteration[i])
for i in range(len(best_solution)):
    sheet1.write(i,2,best_solution[i])
for i in range(len(average_solution_SBM)):
    sheet1.write(i,3,average_solution_SBM[i])
for i in range(len(average_solution_HM)):
```

```
    sheet1.write(i,4,average_solution_HM[i])
for i in range(len(average_solution_PHM)):
    sheet1.write(i,5,average_solution_PHM[i])
for i in range(len(medium_solution_SBM)):
    sheet1.write(i,6,medium_solution_SBM[i])
for i in range(len(medium_solution_HM)):
    sheet1.write(i,7,medium_solution_HM[i])
for i in range(len(medium_solution_PHM)):
    sheet1.write(i,8,medium_solution_PHM[i])
for i in range(len(worst_solution_SBM)):
    sheet1.write(i,9,worst_solution_SBM[i])
for i in range(len(worst_solution_HM)):
    sheet1.write(i,10,worst_solution_HM[i])
for i in range(len(worst_solution_PHM)):
    sheet1.write(i,11,worst_solution_PHM[i])
f.save(file_write)
```

5.4 Testing

The test is divided into two parts. The first part is white-box and black-box testing of the code, as described in chapter 3.The second part is to test the JSP on two machines benchmark problem in the code according to the experimental design in chapter 4.

5.2.1 White box testing

A white box testing is a test of **internal logic structure** in each functional module in the code, and it also applied in unit test. Therefore, the white box test of this experiment is divided into three parts: algorithm function module testing, fitness function module testing, and reading-writing function module testing. The test cases of these module are as follows:

Testing No.1
Testing conditions: Input a 10-bit bit string into the algorithm function EAs with SBM. Count the number of mutations occurring each time. Repeat 1000 times, and then calculate the average number of mutations occurring each time
Expected Results: The average number of mutations occurring is around 1
Actual Results: The average number of mutations occurring is 1.008

Testing No.2
Testing conditions: Input a 10-bit bit string into the algorithm function EAs with SBM.

Count the position of bitstring every time mutation occurs. Repeat 1000 times, and then calculate the average position of mutation occurrence
Expected Results: The average position is around 4.5
Actual Results: The average position is around 4.502

Testing No.3
Testing conditions: Input a 10-bit bit string into the algorithm function AIS with HM. Fitness is not evaluated, and the maximum number of iterations is set to 10. The algorithm is then repeated 1000 times. Print the final bitstring in each time
Expected Results: In 1000 repetitions, bitstring was completely flipped
Actual Results: In 1000 repetitions, bitstring was completely flipped

Testing No.4
Testing conditions: Input a 10-bit bit string into the algorithm function AIS with HM. Fitness is not evaluated, and the maximum number of iterations is set to 10. The algorithm is then repeated 1000 times. In each experiment, the mutation seat occurred in the first iteration was counted. Finally, calculate the average seat when the first iteration occurs
Expected Results: The average seat is around 4.5
Actual Results: The average seat is 4.498

Testing No.5
Testing conditions: Input a 10-bit bit string into the algorithm function AIS with HM. Fitness is not evaluated, and the maximum number of iterations is set to 10. The algorithm is then repeated 1000 times. Calculate the estimated probability after each mutation. At last, the average probability of each evaluation on the mutation list is calculated.
Expected Results: The calculated probability density function is equal to the preset probability density function
Actual Results: The calculated probability density function approximates the preset probability density function

Testing No.6
Testing conditions: Input a 10-bit bit string which has five “1” bit into the fitness function of OneMax benchmark. Then the fitness function runs 1000 times. Count the fitness in each time.
Expected Results: All finesses in every time are 5.
Actual Results: All finesses in every time are 5.

Testing No.7
Testing conditions: Input a 10-bit bit string into the fitness function of job shop scheduling on two machines benchmark, while the correct fitness is known. Then the

fitness function runs 1000 times. Count the fitness in each time.
Expected Results: All finesses in every time are correct.
Actual Results: All finesses in every time are correct.

Testing No.8
Testing conditions: Input a correct path of job shop scheduling on two machines benchmark text file into the reading file function module, Store the contents of the file in a list and output the list.
Expected Results: The file content is consistent with the list content
Actual Results: The file content is consistent with the list content

Testing No.9
Testing conditions: Input the experimental data (based on 10 size JSP on two machines) obtained by the algorithm module and fitness module into writing function module and input a valid path to create the table. Compare the tabular data with the experimental data output in real time in the program
Expected Results: The table data is consistent with the output data in the program
Actual Results: The table data is consistent with the output data in the program

5.2.2 Black box testing

Black box testing is mainly used to test whether the function of the system can be used well. In this project, the OneMax function will be used to test whether the program works. In other words, the testing of JSP on two machines benchmark problem will only be performed if the OneMax benchmark problem testing passes. The test cases of these module are as follows:

Testing No.1
Testing conditions: Test the EAs with SBM on OneMAX benchmark problem. problem size = 100, test times = 100, and the stop condition is fitness = 100
Expected Results: In 100 repeated experiments, the optimal fitness = 100 was obtained
Actual Results: In 100 repeated experiments, the optimal fitness = 100 was obtained

Testing No.2
Testing conditions: Test the AIS with HM on OneMAX benchmark problem. problem size = 100, test times = 100, and the stop condition is fitness = 100
Expected Results: In 100 repeated experiments, the optimal fitness = 100 was obtained
Actual Results: In 100 repeated experiments, the optimal fitness = 100 was obtained

Testing No.3
Testing conditions: Test the AIS with PHM on OneMAX benchmark problem. problem

size = 100, test times = 100, and the stop condition is fitness = 100
Expected Results: In 100 repeated experiments, the optimal fitness = 100 was obtained
Actual Results: In 100 repeated experiments, the optimal fitness = 100 was obtained

Testing No.4
Testing conditions: Test the EAs with SBM on OneMAX benchmark problem. problem size = 100, test times = 100, and the stop condition is fitness = 100. Count the number of iterations when the optimal fitness was reached in 100 replicate experiments was counted. Then calculate the average number of iterations
Expected Results: The average of iteration = 1248
Actual Results: The average of iteration = 1066

Testing No.5
Testing conditions: Test the AIS with HM on OneMAX benchmark problem. problem size = 100, test times = 100, and the stop condition is fitness = 100. Count the number of iterations when the optimal fitness was reached in 100 replicate experiments was counted. Then calculate the average number of iterations
Expected Results: The average of iteration = 48000
Actual Results: The average of iteration = 36358

5.2.3 Job Shop Scheduling on Two Machines Problem Testing

This section is mainly to test the performance of three algorithms in solving JSP on two machines benchmark problem. From the experiment design chapter, there are 4 different max iterations in the testing. Test cases are shown below.

Testing No.1
Testing conditions: Test the EAs with SBM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 300.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate.
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

Testing No.2
Testing conditions: Test the AIS with HM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 300.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst

fitness and success rate.
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

Testing No.3
Testing conditions: Test the AIS with PHM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 300.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate.
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

Testing No.4
Testing conditions: Test the EAs with SBM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 500.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate.
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

Testing No.5
Testing conditions: Test the AIS with HM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 500.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate.
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

Testing No.6
Testing conditions: Test the AIS with PHM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 500.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst

fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

Testing No.7
Testing conditions: Test the EAs with SBM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 1000.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate.
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

Testing No.8
Testing conditions: Test the AIS with HM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 1000.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate.
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

Testing No.9
Testing conditions: Test the AIS with HM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 1000.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate.
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

Testing No.10
Testing conditions: Test the EAs with SBM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 2000.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate.
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

Testing No.11
Testing conditions: Test the AIS with HM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 2000.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate.
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

Testing No.12
Testing conditions: Test the AIS with PHM on JSP on two machines benchmark problem. problem size = 100, test instance = 100, test time in each instance = 100, max iteration = 2000.
Expected Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate.
Actual Results: Get the data of average fitness, medium fitness, best fitness, worst fitness and success rate. Draw scatter diagrams based on fitness result. Draw histogram diagrams based on success rate result.

5.3 Chapter Conclusion

This chapter describes the code implementation of the key functions firstly. Then it introduces the white box testing and black box testing of the project. Finally, it describes the JSP on two machines benchmark problem testing for EAs with SBM, AIS with HM, AIS with PHM.

Chapter 6: Results and discussion

6.1 Results of Experiment

This section discusses the experimental data, which are mainly presented in the form of various tables and charts. What is more, all the original experimental data are put in the appendix.

Firstly, it is worth noting the average fitness of the three algorithms for 100 instances at different maximum iterations, as shown in figures 6.1, 6.2, 6.3 and 6.4. It can be seen that when max iteration = 300, the performance of AIS with HM is significantly weaker than the other two algorithms. Then when max iteration is increased to 500, the performance of AIS with HM is still the worst, but the gap is not as big as when max iteration = 300. When max iteration = 1000, it can be seen from the diagram that the performance of (1+1)EAs with SBM is weaker than AIS with HM in many instances. Finally, when max iteration = 2000, it can be seen from the chart that AIS with HM performs better than EAs with SBM, because EAs with SBM has poorer performance in many instances. In addition to the comparison of the above two algorithms, we can also see that AIS with PHM performs best in the case of four maximum iterations, which always has a smaller fitness gap in each instance.

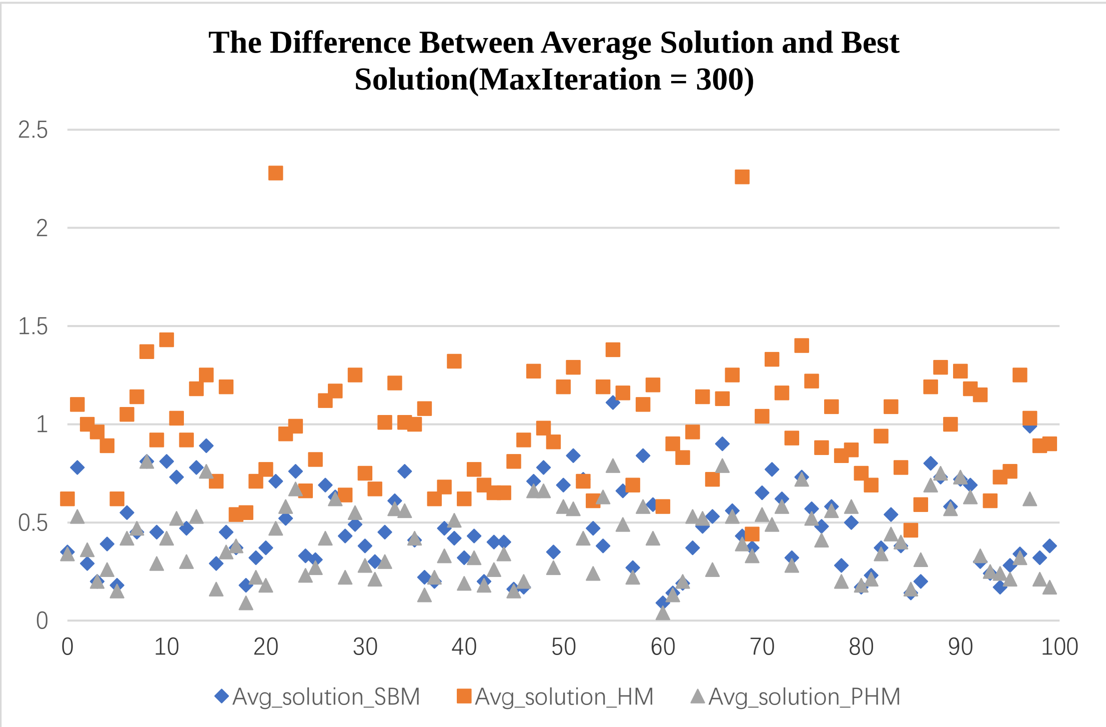

Figure 6.1 The Difference Between Average Solution and Best Solution(MaxIteration = 300)

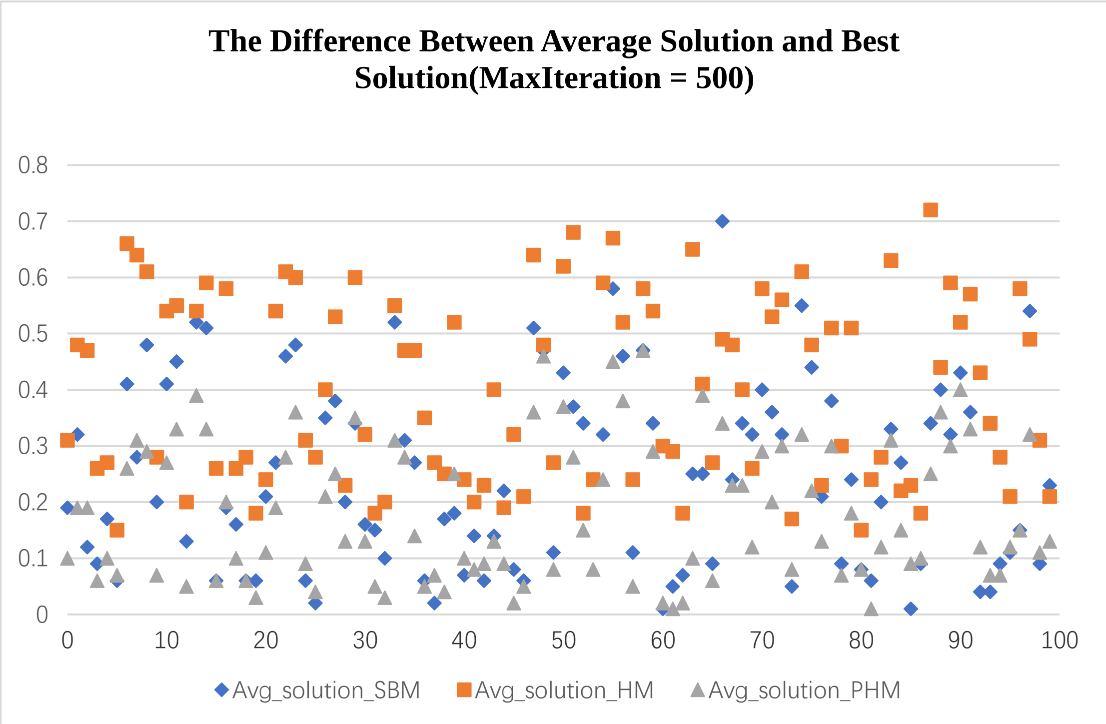

Figure 6.2 The Difference Between Average Solution and Best Solution(MaxIteration = 500)

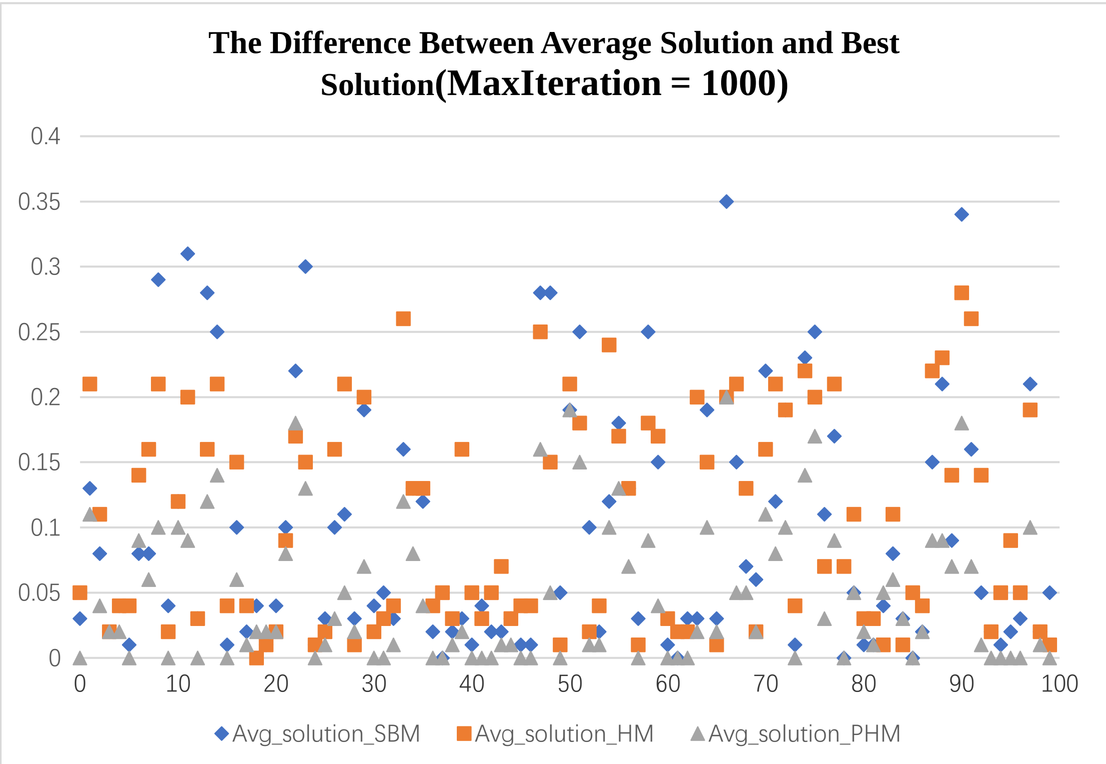

Figure 6.3 The Difference Between Average Solution and Best Solution(MaxIteration = 1000)

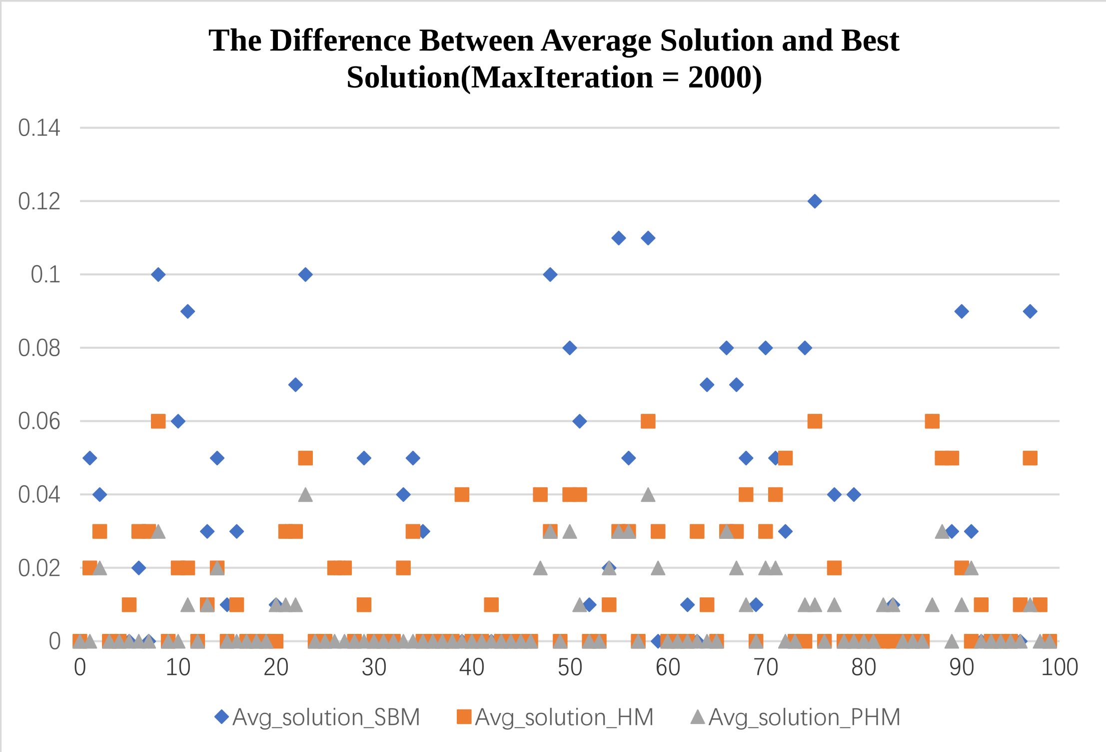

Figure 6.4 The Difference Between Average Solution and Best Solution(MaxIteration = 2000)

In addition, the success rate bar chart also reflects the same performance. The purpose of this experiment is to highlight the ability of the algorithm to escape local optimize. For the JSP on two machines problem, this ability is reflected in how many examples of the algorithm escape from local optimal and achieve global optimal after a certain amount of time of iteration. In other words, in 100 examples of the experiment, 100 repetitions per example, the number of times that the best solution is obtained is a measure of the algorithm's ability to escape the local optimal. Figure XX shows the success rate graph of three experimental algorithms reaching the optimal times under four different maximum iterations. In the case of fewer iterations, AIS with HM cannot better escape from local optimization, but with the increase of the number of iterations, its performance will get better and better until it exceeds the performance of EAs with SBM.AIS with PHM was always the best in the experiment.

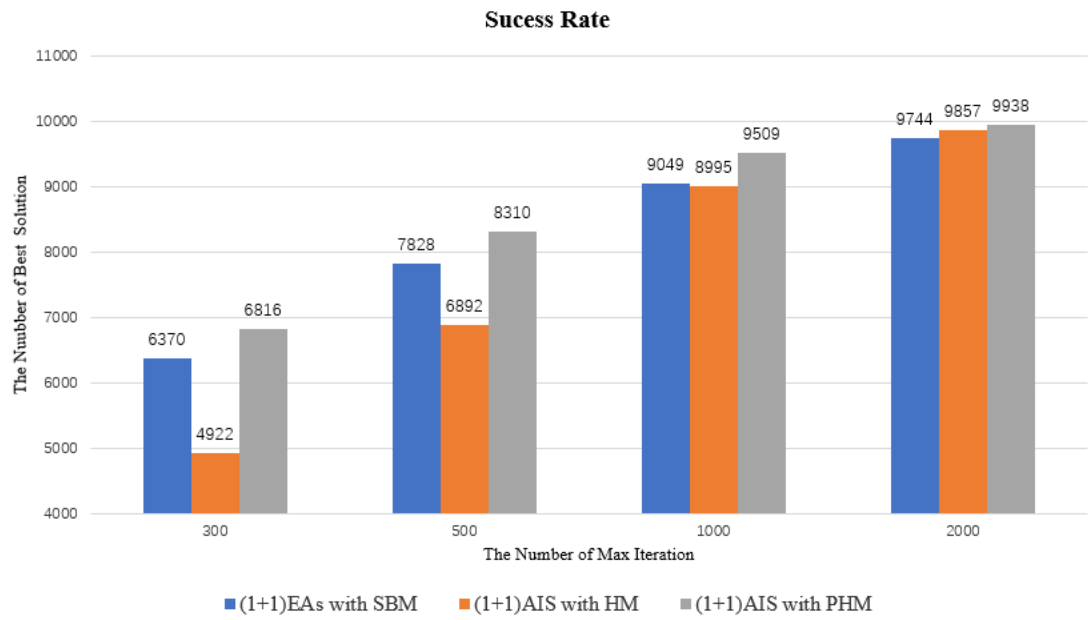

Figure 6.5 Success Rate

6.2 Summary of Experimental Results

From the experimental results, there are two conclusions which have been found:

1. In the benchmark problem of JSP on two machines, only when the maximum number of iterations is large enough can the ability of AIS with HM to escape the local optimum be significantly displayed. Therefore, when the max iteration is small, the ability of HM operator to escape from local optimum does not appear clearly. But with increasing of max iteration, this ability becomes more and more apparent.

2. AIS with PHM performs better than EAs with SBM and AIS with HM regardless of large or small of max iteration. This is because PHM operator reduces the number of invalid fitness evaluations, while PHM retains high mutation rate. This experimental conclusion verifies the theoretical analysis conclusion of a literature[Corus, 2018].

6.3 Further Work

This project compares the ability of SBM, HM and PHM to escape local optimal through a simple combinatorial optimization problem. For the further work, we can further study how to optimize HM and PHM parameters to improve their performance. One parameter optimization is to optimize the maximum number of flips. In this experiment, both HM and PHM operators can completely flips bitstring. In future work, the maximum flips can be reduced to test. Another parameter optimization is to adjust the λ for PHM. In this experiment, λ was fixed at 1/e. The effect of λ adjustment on PHM performance can be tested in future work

In addition, this experiment also has some shortcomings. First, this experiment only analyzes the performance of the algorithm in one combinatorial optimization problem. We should use more practical combinatorial optimization problems to analyze the runtime of these three operators, such as SAT problem. This is because SAT problems can also be tested in the form of bitstring, so the experimental code of this project can be widely reused in future SAT problems testing. Secondly, we should consider more max iteration settings for experiments. In this experiment, only four different max iterations have been set, and only one iteration number setting shows that HM performs better than SBM.

6.4 Chapter Conclusion

This chapter analyzes the experimental data. Firstly, the analysis is presented in the form of scatter diagram and histogram. Then the conclusion of the experiment is summarized, and the experiment objective is completed. Finally, there are some dissuasions about further work and the shortcomings of this project.

Chapter 7: Conclusions

This project mainly analyzes the performance of (1+1)EAs with standard bit mutation operators, (1+1) AIS with hypermutation operators and (1+1) AIS with p-hypermutation operators on job shop scheduling on two machines benchmark. A series of experimental data comparison proves that: 1. (1+1) AIS with hypermutation operators has better performance to escape local optima than (1+1)EAs with standard bit mutation operators when the max iteration is large. 2. (1+1) AIS with p-hypermutation operators has the best performance in the three algorithms. Here is a brief summary of this paper.

Chapter 2: This chapter mainly introduces the background knowledge and related researches. It mainly explains the theoretical knowledge of EAs, AIS and combinatorial optimization problems. Then the application of these two algorithms in combinatorial optimization is highlighted.

Chapter 3: This chapter elaborates on the objective and requirements analysis of this project, including functional requirements and non-functional requirements. Besides, this chapter also explains the risk management plan and ethical issues of the project.

Chapter 4: This chapter mainly introduces the design of algorithm code, including the design of SBM, HM and PHM operators and the design of fitness function. Then this chapter also introduces the design of the experiment, including the design of data collection, and the adjustment of parameters during the experiment.

Chapter 5: This chapter first explains the code implementation of the main functions of the project. The project code is then tested in white box testing and black box testing. Finally, the test cases for the JSP on two machines problem are shown in detail.

Chapter 6: This chapter analyzes the experimental data. Firstly, the analysis is presented in the form of scatter diagram and histogram. Then the conclusion of the experiment is summarized, and the experiment objective is completed. Finally, there are some dissuasions about further work and the shortcomings of this project.

References

Back, T., 1996. Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford university press.

Basu, M., 2011. Artificial immune system for fixed head hydrothermal power system. Energy, 36(1), pp.606-612.

Bertsekas, D.P., Tsitsiklis, J.N. and Wu, C., 1997. Rollout algorithms for combinatorial optimization. Journal of Heuristics, 3(3), pp.245-262.

Chou, P.H., 2002. Algorithm education in Python. Proceedings of Python, 10, p.2.

Cantú-Paz, E., 2002, July. On random numbers and the performance of genetic algorithms. In Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation (pp. 311-318). Morgan Kaufmann Publishers Inc.

Cook, S.A., 1971, May. The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on Theory of computing (pp. 151-158). ACM.

Corus, D., Oliveto, P.S. and Yazdani, D., 2018, September. Artificial immune systems can find arbitrarily good approximations for the NP-hard partition problem. In International Conference on Parallel Problem Solving from Nature (pp. 16-28). Springer, Cham.

Corus, D., Oliveto, P.S. and Yazdani, D., 2018, September. Fast artificial immune systems. In International Conference on Parallel Problem Solving from Nature (pp. 67-78). Springer, Cham.

Cutello, V., Nicosia, G. and Pavone, M., 2004, September. Exploring the capability of immune algorithms: A characterization of hypermutation operators. In International Conference on Artificial Immune Systems (pp. 263-276). Springer, Berlin, Heidelberg.

D. Corus, P. S. Oliveto, D. Yazdani, On the runtime analysis of the Opt-IA artificial immune system, in: Proc. of GECCO 2017, 2017, pp. 83–90

De Castro, L. N., & Von Zuben, F. J. (2002). Learning and optimization using the clonal selection principle. IEEE transactions on evolutionary computation, 6(3), 239-251.

Deng, W., Zhao, H., Zou, L., Li, G., Yang, X. and Wu, D., 2017. A novel collaborative optimization algorithm in solving complex optimization problems. Soft

Computing, 21(15), pp.4387-4398.

Du, Hai-Feng, Li-Cheng Jiao, and Sun-An Wang. "Clonal operator and antibody clone algorithms." Proceedings. International Conference on Machine Learning and Cybernetics. Vol. 1. IEEE, 2002.

Farmer, J.D., Packard, N.H. and Perelson, A.S., 1986. The immune system, adaptation, and machine learning. Physica D: Nonlinear Phenomena, 22(1-3), pp.187-204.

Fortin, F.A., Rainville, F.M.D., Gardner, M.A., Parizeau, M. and Gagné, C., 2012. DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research, 13(Jul), pp.2171-2175.

Janeway, C.A., Travers, P., Walport, M. and Shlomchik, M., 1996. Immunobiology: the immune system in health and disease (Vol. 7, p. 26). London: Current Biology.

Kephart, J. O. (1994). "A biologically inspired immune system for computers". Proceedings of Artificial Life IV: The Fourth International Workshop on the Synthesis and Simulation of Living Systems. MIT Press. pp. 130–139.

Krippendorff, Klaus. "Combinatorial Explosion". Web Dictionary of Cybernetics and Systems. PRINCIPIA CYBERNETICA WEB. Retrieved 29 November 2010.

Lin, S.W. and Ying, K.C., 2013. Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm. Omega, 41(2), pp.383-389.

Oda, T. and White, T., 2005, August. Immunity from spam: An analysis of an artificial immune system for junk email detection. In International conference on artificial immune systems (pp. 276-289). Springer, Berlin, Heidelberg.

Papadimitriou, C.H. and Steiglitz, K., 1998. 6.1 The Max-Flow, Min-Cut Theorem. Combinatorial optimization: Algorithms and complexity, Dover, pp.120-128.

Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+ 1)evolutionary algorithm. Theor. Comp. Sci., 276(1-2):51–81, 2002

Thomas Jansen and Christine Zarges. Analyzing different variants of immune inspired somatic contiguous hypermutations. Theor. Comp. Sci., 412(6):517 – 533,2011

Appendices

Instance	Max iteration	Best solution	Average solution(SBM)	Average solution(HM)	Average solution(PHM)
0	300	2691	2691.35	2691.62	2691.34
1	300	2439	2439.78	2440.1	2439.53
2	300	2532	2532.29	2533	2532.36
3	300	2599	2599.2	2599.96	2599.2
4	300	2639	2639.39	2639.89	2639.26
5	300	2375	2375.18	2375.62	2375.15
6	300	2571	2571.55	2572.05	2571.42
7	300	2490	2490.45	2491.14	2490.47
8	300	2581	2581.81	2582.37	2581.81
9	300	2575	2575.45	2575.92	2575.29
10	300	2298	2298.81	2299.43	2298.42
11	300	2554	2554.73	2555.03	2554.52
12	300	2535	2535.47	2535.92	2535.3
13	300	2441	2441.78	2442.18	2441.53
14	300	2540	2540.89	2541.25	2540.76
15	300	2457	2457.29	2457.71	2457.16
16	300	2374	2374.45	2375.19	2374.35
17	300	2468	2468.37	2468.54	2468.38
18	300	2308	2308.18	2308.55	2308.09
19	300	2409	2409.32	2409.71	2409.22
20	300	2560	2560.37	2560.77	2560.18
21	300	2487	2487.71	2489.28	2487.47
22	300	2587	2587.52	2587.95	2587.58
23	300	2344	2344.76	2344.99	2344.67
24	300	2572	2572.33	2572.66	2572.23
25	300	2493	2493.31	2493.82	2493.27
26	300	2418	2418.69	2419.12	2418.42
27	300	2388	2388.63	2389.17	2388.62
28	300	2441	2441.43	2441.64	2441.22
29	300	2516	2516.49	2517.25	2516.55
30	300	2582	2582.38	2582.75	2582.28
31	300	2335	2335.3	2335.67	2335.21
32	300	2591	2591.45	2592.01	2591.3
33	300	2560	2560.61	2561.21	2560.57
34	300	2308	2308.76	2309.01	2308.56
35	300	2226	2226.41	2227	2226.42
36	300	2499	2499.22	2500.08	2499.13
37	300	2445	2445.2	2445.62	2445.22

38	300	2502	2502.47	2502.68	2502.33
39	300	2499	2499.42	2500.32	2499.51
40	300	2514	2514.32	2514.62	2514.19
41	300	2588	2588.43	2588.77	2588.32
42	300	2493	2493.2	2493.69	2493.18
43	300	2645	2645.4	2645.65	2645.26
44	300	2682	2682.4	2682.65	2682.34
45	300	2617	2617.16	2617.81	2617.15
46	300	2594	2594.17	2594.92	2594.2
47	300	2627	2627.71	2628.27	2627.66
48	300	2440	2440.78	2440.98	2440.66
49	300	2515	2515.35	2515.91	2515.27
50	300	2670	2670.69	2671.19	2670.58
51	300	2522	2522.84	2523.29	2522.57
52	300	2527	2527.72	2527.71	2527.42
53	300	2513	2513.47	2513.61	2513.24
54	300	2611	2611.38	2612.19	2611.63
55	300	2818	2819.11	2819.38	2818.79
56	300	2635	2635.66	2636.16	2635.49
57	300	2453	2453.27	2453.69	2453.22
58	300	2724	2724.84	2725.1	2724.58
59	300	2484	2484.59	2485.2	2484.42
60	300	2218	2218.09	2218.58	2218.04
61	300	2450	2450.14	2450.9	2450.13
62	300	2462	2462.19	2462.83	2462.2
63	300	2550	2550.37	2550.96	2550.53
64	300	2325	2325.48	2326.14	2325.52
65	300	2472	2472.53	2472.72	2472.26
66	300	2591	2591.9	2592.13	2591.79
67	300	2463	2463.56	2464.25	2463.53
68	300	2248	2248.43	2250.26	2248.39
69	300	2441	2441.37	2441.44	2441.33
70	300	2502	2502.65	2503.04	2502.54
71	300	2638	2638.77	2639.33	2638.49
72	300	2401	2401.62	2402.16	2401.58
73	300	2449	2449.32	2449.93	2449.28
74	300	2750	2750.73	2751.4	2750.72
75	300	2514	2514.57	2515.22	2514.52
76	300	2425	2425.48	2425.88	2425.41
77	300	2723	2723.58	2724.09	2723.56
78	300	2474	2474.28	2474.84	2474.2
79	300	2224	2224.5	2224.87	2224.58

80	300	2534	2534.17	2534.75	2534.18
81	300	2656	2656.23	2656.69	2656.21
82	300	2770	2770.37	2770.94	2770.34
83	300	2420	2420.54	2421.09	2420.44
84	300	2467	2467.38	2467.78	2467.4
85	300	2695	2695.14	2695.46	2695.16
86	300	2639	2639.2	2639.59	2639.31
87	300	2678	2678.8	2679.19	2678.69
88	300	2745	2745.73	2746.29	2745.75
89	300	2718	2718.58	2719	2718.57
90	300	2520	2520.72	2521.27	2520.73
91	300	2539	2539.69	2540.18	2539.63
92	300	2432	2432.3	2433.15	2432.33
93	300	2434	2434.24	2434.61	2434.25
94	300	2242	2242.17	2242.73	2242.24
95	300	2618	2618.28	2618.76	2618.21
96	300	2495	2495.34	2496.25	2495.32
97	300	2645	2645.99	2646.03	2645.62
98	300	2628	2628.32	2628.89	2628.21
99	300	2301	2301.38	2301.9	2301.17

Figure A-1: Average Fitness Result(MaxIteration = 300)

Instance	Max iteration	Best solution	Medium solution(SBM)	Medium solution(HM)	Medium solution(PHM)
0	300	2691	2691	2691	2691
1	300	2439	2440	2440	2439
2	300	2532	2532	2533	2532
3	300	2599	2599	2599	2599
4	300	2639	2639	2639	2639
5	300	2375	2375	2375	2375
6	300	2571	2571	2572	2571
7	300	2490	2490	2491	2490
8	300	2581	2582	2582	2582
9	300	2575	2575	2575	2575
10	300	2298	2299	2299	2298
11	300	2554	2555	2555	2554
12	300	2535	2535	2535	2535
13	300	2441	2442	2442	2441
14	300	2540	2541	2541	2541
15	300	2457	2457	2457	2457
16	300	2374	2374	2375	2374

17	300	2468	2468	2468	2468
18	300	2308	2308	2308	2308
19	300	2409	2409	2409	2409
20	300	2560	2560	2560	2560
21	300	2487	2487.5	2488	2487
22	300	2587	2587	2588	2588
23	300	2344	2345	2345	2345
24	300	2572	2572	2572	2572
25	300	2493	2493	2493	2493
26	300	2418	2418	2419	2418
27	300	2388	2388	2389	2388
28	300	2441	2441	2441	2441
29	300	2516	2516	2517	2516
30	300	2582	2582	2582	2582
31	300	2335	2335	2335	2335
32	300	2591	2591	2591	2591
33	300	2560	2560	2561	2560
34	300	2308	2309	2309	2308
35	300	2226	2226	2227	2226
36	300	2499	2499	2499	2499
37	300	2445	2445	2445	2445
38	300	2502	2502	2502	2502
39	300	2499	2499	2500	2499
40	300	2514	2514	2514	2514
41	300	2588	2588	2588	2588
42	300	2493	2493	2493	2493
43	300	2645	2645	2645	2645
44	300	2682	2682	2682	2682
45	300	2617	2617	2617	2617
46	300	2594	2594	2594	2594
47	300	2627	2628	2628	2628
48	300	2440	2441	2441	2441
49	300	2515	2515	2515	2515
50	300	2670	2671	2671	2670
51	300	2522	2523	2523	2522
52	300	2527	2527	2527	2527
53	300	2513	2513	2513	2513
54	300	2611	2611	2612	2612
55	300	2818	2819	2819	2819
56	300	2635	2635	2636	2635
57	300	2453	2453	2453	2453
58	300	2724	2725	2725	2724

59	300	2484	2485	2485	2484
60	300	2218	2218	2218	2218
61	300	2450	2450	2450	2450
62	300	2462	2462	2462	2462
63	300	2550	2550	2551	2550
64	300	2325	2325	2326	2325
65	300	2472	2472	2472	2472
66	300	2591	2592	2592	2592
67	300	2463	2463	2464	2463
68	300	2248	2248	2249	2248
69	300	2441	2441	2441	2441
70	300	2502	2503	2503	2502
71	300	2638	2638	2639	2638
72	300	2401	2401	2402	2401
73	300	2449	2449	2449	2449
74	300	2750	2751	2751	2751
75	300	2514	2515	2515	2514.5
76	300	2425	2425	2425	2425
77	300	2723	2723	2724	2723
78	300	2474	2474	2474	2474
79	300	2224	2224	2225	2224
80	300	2534	2534	2534	2534
81	300	2656	2656	2656	2656
82	300	2770	2770	2770	2770
83	300	2420	2420	2421	2420
84	300	2467	2467	2467	2467
85	300	2695	2695	2695	2695
86	300	2639	2639	2639	2639
87	300	2678	2679	2679	2679
88	300	2745	2746	2746	2746
89	300	2718	2718	2719	2718
90	300	2520	2521	2521	2521
91	300	2539	2540	2540	2539
92	300	2432	2432	2433	2432
93	300	2434	2434	2434	2434
94	300	2242	2242	2242	2242
95	300	2618	2618	2618	2618
96	300	2495	2495	2496	2495
97	300	2645	2646	2646	2645
98	300	2628	2628	2628	2628
99	300	2301	2301	2301	2301

Figure A-2: Medium Fitness Result(MaxIteration = 300)

Instance	Max iteration	Best solution	Worst solution(SBM)	Worst solution(HM)	Worst solution(PHM)
0	300	2691	2694	2697	2693
1	300	2439	2442	2446	2443
2	300	2532	2534	2538	2534
3	300	2599	2602	2609	2602
4	300	2639	2641	2647	2642
5	300	2375	2377	2379	2377
6	300	2571	2573	2579	2574
7	300	2490	2492	2496	2493
8	300	2581	2584	2601	2584
9	300	2575	2578	2582	2578
10	300	2298	2301	2307	2300
11	300	2554	2557	2562	2559
12	300	2535	2540	2542	2539
13	300	2441	2445	2449	2444
14	300	2540	2545	2548	2543
15	300	2457	2460	2465	2459
16	300	2374	2377	2380	2376
17	300	2468	2470	2472	2471
18	300	2308	2311	2314	2309
19	300	2409	2413	2416	2412
20	300	2560	2563	2565	2562
21	300	2487	2490	2602	2489
22	300	2587	2590	2595	2589
23	300	2344	2347	2350	2348
24	300	2572	2575	2579	2575
25	300	2493	2495	2499	2496
26	300	2418	2422	2432	2421
27	300	2388	2391	2395	2391
28	300	2441	2445	2451	2444
29	300	2516	2518	2527	2519
30	300	2582	2585	2592	2585
31	300	2335	2337	2341	2338
32	300	2591	2596	2598	2594
33	300	2560	2564	2567	2564
34	300	2308	2311	2315	2312
35	300	2226	2229	2235	2228
36	300	2499	2501	2521	2501
37	300	2445	2448	2449	2447
38	300	2502	2505	2506	2505

39	300	2499	2502	2506	2503
40	300	2514	2517	2518	2516
41	300	2588	2590	2598	2591
42	300	2493	2495	2497	2495
43	300	2645	2648	2650	2649
44	300	2682	2686	2688	2685
45	300	2617	2619	2628	2619
46	300	2594	2596	2601	2596
47	300	2627	2629	2634	2629
48	300	2440	2443	2447	2443
49	300	2515	2519	2534	2517
50	300	2670	2677	2677	2674
51	300	2522	2526	2552	2525
52	300	2527	2531	2538	2531
53	300	2513	2516	2519	2516
54	300	2611	2613	2616	2616
55	300	2818	2824	2837	2823
56	300	2635	2639	2643	2638
57	300	2453	2457	2459	2455
58	300	2724	2729	2730	2727
59	300	2484	2487	2492	2487
60	300	2218	2221	2222	2219
61	300	2450	2452	2462	2452
62	300	2462	2464	2498	2464
63	300	2550	2553	2559	2553
64	300	2325	2328	2331	2328
65	300	2472	2475	2479	2475
66	300	2591	2594	2601	2596
67	300	2463	2465	2474	2465
68	300	2248	2251	2323	2252
69	300	2441	2444	2447	2443
70	300	2502	2504	2509	2506
71	300	2638	2643	2648	2641
72	300	2401	2404	2407	2405
73	300	2449	2452	2457	2451
74	300	2750	2754	2762	2754
75	300	2514	2516	2524	2516
76	300	2425	2428	2436	2428
77	300	2723	2726	2728	2726
78	300	2474	2476	2482	2476
79	300	2224	2227	2230	2227
80	300	2534	2536	2542	2536

81	300	2656	2660	2662	2659
82	300	2770	2772	2778	2773
83	300	2420	2423	2430	2422
84	300	2467	2471	2478	2470
85	300	2695	2697	2701	2697
86	300	2639	2641	2645	2644
87	300	2678	2683	2684	2681
88	300	2745	2748	2768	2748
89	300	2718	2722	2723	2722
90	300	2520	2524	2529	2523
91	300	2539	2543	2545	2544
92	300	2432	2437	2437	2435
93	300	2434	2436	2441	2439
94	300	2242	2244	2252	2245
95	300	2618	2621	2623	2621
96	300	2495	2499	2501	2498
97	300	2645	2653	2650	2648
98	300	2628	2631	2639	2630
99	300	2301	2303	2326	2302

Figure A-3: Worst Fitness Result(MaxIteration = 300)

Instance	Max iteration	Best solution	Average solution(SBM)	Average solution(HM)	Average solution(PHM)
0	500	2691	2691.19	2691.31	2691.1
1	500	2439	2439.32	2439.48	2439.19
2	500	2532	2532.12	2532.47	2532.19
3	500	2599	2599.09	2599.26	2599.06
4	500	2639	2639.17	2639.27	2639.1
5	500	2375	2375.06	2375.15	2375.07
6	500	2571	2571.41	2571.66	2571.26
7	500	2490	2490.28	2490.64	2490.31
8	500	2581	2581.48	2581.61	2581.29
9	500	2575	2575.2	2575.28	2575.07
10	500	2298	2298.41	2298.54	2298.27
11	500	2554	2554.45	2554.55	2554.33
12	500	2535	2535.13	2535.2	2535.05
13	500	2441	2441.52	2441.54	2441.39
14	500	2540	2540.51	2540.59	2540.33
15	500	2457	2457.06	2457.26	2457.06
16	500	2374	2374.19	2374.58	2374.2
17	500	2468	2468.16	2468.26	2468.1
18	500	2308	2308.06	2308.28	2308.06

19	500	2409	2409.06	2409.18	2409.03
20	500	2560	2560.21	2560.24	2560.11
21	500	2487	2487.27	2487.54	2487.19
22	500	2587	2587.46	2587.61	2587.28
23	500	2344	2344.48	2344.6	2344.36
24	500	2572	2572.06	2572.31	2572.09
25	500	2493	2493.02	2493.28	2493.04
26	500	2418	2418.35	2418.4	2418.21
27	500	2388	2388.38	2388.53	2388.25
28	500	2441	2441.2	2441.23	2441.13
29	500	2516	2516.34	2516.6	2516.35
30	500	2582	2582.16	2582.32	2582.13
31	500	2335	2335.15	2335.18	2335.05
32	500	2591	2591.1	2591.2	2591.03
33	500	2560	2560.52	2560.55	2560.31
34	500	2308	2308.31	2308.47	2308.28
35	500	2226	2226.27	2226.47	2226.14
36	500	2499	2499.06	2499.35	2499.05
37	500	2445	2445.02	2445.27	2445.07
38	500	2502	2502.17	2502.25	2502.04
39	500	2499	2499.18	2499.52	2499.25
40	500	2514	2514.07	2514.24	2514.1
41	500	2588	2588.14	2588.2	2588.08
42	500	2493	2493.06	2493.23	2493.09
43	500	2645	2645.14	2645.4	2645.13
44	500	2682	2682.22	2682.19	2682.09
45	500	2617	2617.08	2617.32	2617.02
46	500	2594	2594.06	2594.21	2594.05
47	500	2627	2627.51	2627.64	2627.36
48	500	2440	2440.47	2440.48	2440.46
49	500	2515	2515.11	2515.27	2515.08
50	500	2670	2670.43	2670.62	2670.37
51	500	2522	2522.37	2522.68	2522.28
52	500	2527	2527.34	2527.18	2527.15
53	500	2513	2513.24	2513.24	2513.08
54	500	2611	2611.32	2611.59	2611.24
55	500	2818	2818.58	2818.67	2818.45
56	500	2635	2635.46	2635.52	2635.38
57	500	2453	2453.11	2453.24	2453.05
58	500	2724	2724.47	2724.58	2724.47
59	500	2484	2484.34	2484.54	2484.29
60	500	2218	2218.01	2218.3	2218.02

61	500	2450	2450.05	2450.29	2450.01
62	500	2462	2462.07	2462.18	2462.02
63	500	2550	2550.25	2550.65	2550.1
64	500	2325	2325.25	2325.41	2325.39
65	500	2472	2472.09	2472.27	2472.06
66	500	2591	2591.7	2591.49	2591.34
67	500	2463	2463.24	2463.48	2463.23
68	500	2248	2248.34	2248.4	2248.23
69	500	2441	2441.32	2441.26	2441.12
70	500	2502	2502.4	2502.58	2502.29
71	500	2638	2638.36	2638.53	2638.2
72	500	2401	2401.32	2401.56	2401.3
73	500	2449	2449.05	2449.17	2449.08
74	500	2750	2750.55	2750.61	2750.32
75	500	2514	2514.44	2514.48	2514.22
76	500	2425	2425.21	2425.23	2425.13
77	500	2723	2723.38	2723.51	2723.3
78	500	2474	2474.09	2474.3	2474.07
79	500	2224	2224.24	2224.51	2224.18
80	500	2534	2534.08	2534.15	2534.08
81	500	2656	2656.06	2656.24	2656.01
82	500	2770	2770.2	2770.28	2770.12
83	500	2420	2420.33	2420.63	2420.31
84	500	2467	2467.27	2467.22	2467.15
85	500	2695	2695.01	2695.23	2695.09
86	500	2639	2639.09	2639.18	2639.1
87	500	2678	2678.34	2678.72	2678.25
88	500	2745	2745.4	2745.44	2745.36
89	500	2718	2718.32	2718.59	2718.3
90	500	2520	2520.43	2520.52	2520.4
91	500	2539	2539.36	2539.57	2539.33
92	500	2432	2432.04	2432.43	2432.12
93	500	2434	2434.04	2434.34	2434.07
94	500	2242	2242.09	2242.28	2242.07
95	500	2618	2618.11	2618.21	2618.12
96	500	2495	2495.15	2495.58	2495.15
97	500	2645	2645.54	2645.49	2645.32
98	500	2628	2628.09	2628.31	2628.11
99	500	2301	2301.23	2301.21	2301.13

Figure A-4: Average Fitness Result(MaxIteration = 500)

Instance	Max iteration	Best solution	Medium solution(SBM)	Medium solution(HM)	Medium solution(PHM)
0	500	2691	2691	2691	2691
1	500	2439	2439	2439	2439
2	500	2532	2532	2532	2532
3	500	2599	2599	2599	2599
4	500	2639	2639	2639	2639
5	500	2375	2375	2375	2375
6	500	2571	2571	2572	2571
7	500	2490	2490	2490	2490
8	500	2581	2581	2581	2581
9	500	2575	2575	2575	2575
10	500	2298	2298	2298	2298
11	500	2554	2554	2554	2554
12	500	2535	2535	2535	2535
13	500	2441	2441	2441	2441
14	500	2540	2540	2540	2540
15	500	2457	2457	2457	2457
16	500	2374	2374	2374	2374
17	500	2468	2468	2468	2468
18	500	2308	2308	2308	2308
19	500	2409	2409	2409	2409
20	500	2560	2560	2560	2560
21	500	2487	2487	2487	2487
22	500	2587	2587	2588	2587
23	500	2344	2344	2344	2344
24	500	2572	2572	2572	2572
25	500	2493	2493	2493	2493
26	500	2418	2418	2418	2418
27	500	2388	2388	2388	2388
28	500	2441	2441	2441	2441
29	500	2516	2516	2516	2516
30	500	2582	2582	2582	2582
31	500	2335	2335	2335	2335
32	500	2591	2591	2591	2591
33	500	2560	2560	2560	2560
34	500	2308	2308	2308	2308
35	500	2226	2226	2226	2226
36	500	2499	2499	2499	2499
37	500	2445	2445	2445	2445
38	500	2502	2502	2502	2502
39	500	2499	2499	2499	2499

40	500	2514	2514	2514	2514
41	500	2588	2588	2588	2588
42	500	2493	2493	2493	2493
43	500	2645	2645	2645	2645
44	500	2682	2682	2682	2682
45	500	2617	2617	2617	2617
46	500	2594	2594	2594	2594
47	500	2627	2627	2627.5	2627
48	500	2440	2440	2440	2440
49	500	2515	2515	2515	2515
50	500	2670	2670	2671	2670
51	500	2522	2522	2522.5	2522
52	500	2527	2527	2527	2527
53	500	2513	2513	2513	2513
54	500	2611	2611	2611	2611
55	500	2818	2818	2818	2818
56	500	2635	2635	2635	2635
57	500	2453	2453	2453	2453
58	500	2724	2724	2724	2724
59	500	2484	2484	2484	2484
60	500	2218	2218	2218	2218
61	500	2450	2450	2450	2450
62	500	2462	2462	2462	2462
63	500	2550	2550	2550.5	2550
64	500	2325	2325	2325	2325
65	500	2472	2472	2472	2472
66	500	2591	2591	2591	2591
67	500	2463	2463	2463	2463
68	500	2248	2248	2248	2248
69	500	2441	2441	2441	2441
70	500	2502	2502	2502	2502
71	500	2638	2638	2638	2638
72	500	2401	2401	2401	2401
73	500	2449	2449	2449	2449
74	500	2750	2751	2750	2750
75	500	2514	2514	2514	2514
76	500	2425	2425	2425	2425
77	500	2723	2723	2723	2723
78	500	2474	2474	2474	2474
79	500	2224	2224	2224	2224
80	500	2534	2534	2534	2534
81	500	2656	2656	2656	2656

82	500	2770	2770	2770	2770
83	500	2420	2420	2421	2420
84	500	2467	2467	2467	2467
85	500	2695	2695	2695	2695
86	500	2639	2639	2639	2639
87	500	2678	2678	2679	2678
88	500	2745	2745	2745	2745
89	500	2718	2718	2718	2718
90	500	2520	2520	2520	2520
91	500	2539	2539	2539	2539
92	500	2432	2432	2432	2432
93	500	2434	2434	2434	2434
94	500	2242	2242	2242	2242
95	500	2618	2618	2618	2618
96	500	2495	2495	2495	2495
97	500	2645	2645	2645	2645
98	500	2628	2628	2628	2628
99	500	2301	2301	2301	2301

Figure A-5: Medium Fitness Result(MaxIteration = 500)

Instance	Max iteration	Best solution	Worst solution(SBM)	Worst solution(HM)	Worst solution(PHM)
0	500	2691	2693	2694	2692
1	500	2439	2442	2442	2441
2	500	2532	2533	2534	2534
3	500	2599	2600	2601	2600
4	500	2639	2641	2645	2641
5	500	2375	2376	2377	2376
6	500	2571	2573	2574	2573
7	500	2490	2491	2493	2491
8	500	2581	2584	2586	2583
9	500	2575	2577	2577	2576
10	500	2298	2301	2302	2300
11	500	2554	2557	2557	2557
12	500	2535	2539	2539	2537
13	500	2441	2444	2444	2444
14	500	2540	2543	2544	2542
15	500	2457	2458	2460	2458
16	500	2374	2376	2377	2376
17	500	2468	2470	2470	2469
18	500	2308	2309	2311	2309
19	500	2409	2410	2411	2410

20	500	2560	2562	2564	2561
21	500	2487	2489	2490	2488
22	500	2587	2589	2590	2588
23	500	2344	2346	2348	2346
24	500	2572	2573	2574	2573
25	500	2493	2494	2496	2494
26	500	2418	2420	2421	2420
27	500	2388	2391	2394	2389
28	500	2441	2443	2444	2443
29	500	2516	2518	2519	2518
30	500	2582	2584	2585	2584
31	500	2335	2336	2337	2336
32	500	2591	2593	2594	2592
33	500	2560	2563	2564	2562
34	500	2308	2310	2312	2310
35	500	2226	2228	2229	2227
36	500	2499	2500	2502	2500
37	500	2445	2446	2449	2447
38	500	2502	2505	2504	2503
39	500	2499	2500	2502	2501
40	500	2514	2515	2517	2516
41	500	2588	2589	2591	2590
42	500	2493	2494	2495	2495
43	500	2645	2647	2650	2647
44	500	2682	2685	2684	2684
45	500	2617	2618	2621	2618
46	500	2594	2596	2595	2595
47	500	2627	2629	2630	2630
48	500	2440	2442	2445	2442
49	500	2515	2516	2518	2516
50	500	2670	2672	2674	2671
51	500	2522	2524	2526	2524
52	500	2527	2530	2529	2529
53	500	2513	2516	2517	2514
54	500	2611	2612	2615	2612
55	500	2818	2821	2822	2820
56	500	2635	2637	2638	2639
57	500	2453	2455	2455	2454
58	500	2724	2727	2728	2726
59	500	2484	2486	2487	2485
60	500	2218	2219	2225	2219
61	500	2450	2451	2454	2451

62	500	2462	2463	2465	2463
63	500	2550	2552	2554	2552
64	500	2325	2326	2328	2328
65	500	2472	2473	2476	2473
66	500	2591	2594	2594	2593
67	500	2463	2464	2469	2465
68	500	2248	2250	2250	2249
69	500	2441	2443	2444	2443
70	500	2502	2504	2509	2504
71	500	2638	2641	2642	2639
72	500	2401	2404	2405	2403
73	500	2449	2450	2451	2450
74	500	2750	2752	2755	2752
75	500	2514	2515	2517	2515
76	500	2425	2427	2428	2427
77	500	2723	2725	2727	2725
78	500	2474	2475	2479	2475
79	500	2224	2226	2226	2225
80	500	2534	2535	2536	2535
81	500	2656	2657	2660	2657
82	500	2770	2772	2773	2772
83	500	2420	2423	2423	2423
84	500	2467	2469	2470	2469
85	500	2695	2696	2699	2697
86	500	2639	2641	2641	2640
87	500	2678	2681	2684	2680
88	500	2745	2747	2749	2747
89	500	2718	2720	2720	2720
90	500	2520	2522	2524	2523
91	500	2539	2543	2544	2541
92	500	2432	2433	2434	2434
93	500	2434	2436	2440	2436
94	500	2242	2243	2246	2243
95	500	2618	2620	2621	2620
96	500	2495	2497	2500	2498
97	500	2645	2648	2648	2647
98	500	2628	2630	2631	2630
99	500	2301	2303	2303	2303

Figure A-6: Worst Fitness Result(MaxIteration = 500)

Instance	Max iteration	Best solution	Average solution(SBM)	Average solution(HM)	Average solution(PHM)

0	1000	2691	2691.03	2691.05	2691
1	1000	2439	2439.13	2439.21	2439.11
2	1000	2532	2532.08	2532.11	2532.04
3	1000	2599	2599.02	2599.02	2599.02
4	1000	2639	2639.04	2639.04	2639.02
5	1000	2375	2375.01	2375.04	2375
6	1000	2571	2571.08	2571.14	2571.09
7	1000	2490	2490.08	2490.16	2490.06
8	1000	2581	2581.29	2581.21	2581.1
9	1000	2575	2575.04	2575.02	2575
10	1000	2298	2298.12	2298.12	2298.1
11	1000	2554	2554.31	2554.2	2554.09
12	1000	2535	2535.03	2535.03	2535
13	1000	2441	2441.28	2441.16	2441.12
14	1000	2540	2540.25	2540.21	2540.14
15	1000	2457	2457.01	2457.04	2457
16	1000	2374	2374.1	2374.15	2374.06
17	1000	2468	2468.02	2468.04	2468.01
18	1000	2308	2308.04	2308	2308.02
19	1000	2409	2409.01	2409.01	2409.02
20	1000	2560	2560.04	2560.02	2560.02
21	1000	2487	2487.1	2487.09	2487.08
22	1000	2587	2587.22	2587.17	2587.18
23	1000	2344	2344.3	2344.15	2344.13
24	1000	2572	2572.01	2572.01	2572
25	1000	2493	2493.03	2493.02	2493.01
26	1000	2418	2418.1	2418.16	2418.03
27	1000	2388	2388.11	2388.21	2388.05
28	1000	2441	2441.03	2441.01	2441.02
29	1000	2516	2516.19	2516.2	2516.07
30	1000	2582	2582.04	2582.02	2582
31	1000	2335	2335.05	2335.03	2335
32	1000	2591	2591.03	2591.04	2591.01
33	1000	2560	2560.16	2560.26	2560.12
34	1000	2308	2308.13	2308.13	2308.08
35	1000	2226	2226.12	2226.13	2226.04
36	1000	2499	2499.02	2499.04	2499
37	1000	2445	2445	2445.05	2445
38	1000	2502	2502.02	2502.03	2502.01
39	1000	2499	2499.03	2499.16	2499.02
40	1000	2514	2514.01	2514.05	2514
41	1000	2588	2588.04	2588.03	2588

42	1000	2493	2493.02	2493.05	2493
43	1000	2645	2645.02	2645.07	2645.01
44	1000	2682	2682.03	2682.03	2682.01
45	1000	2617	2617.01	2617.04	2617
46	1000	2594	2594.01	2594.04	2594
47	1000	2627	2627.28	2627.25	2627.16
48	1000	2440	2440.28	2440.15	2440.05
49	1000	2515	2515.05	2515.01	2515
50	1000	2670	2670.19	2670.21	2670.19
51	1000	2522	2522.25	2522.18	2522.15
52	1000	2527	2527.1	2527.02	2527.01
53	1000	2513	2513.02	2513.04	2513.01
54	1000	2611	2611.12	2611.24	2611.1
55	1000	2818	2818.18	2818.17	2818.13
56	1000	2635	2635.13	2635.13	2635.07
57	1000	2453	2453.03	2453.01	2453
58	1000	2724	2724.25	2724.18	2724.09
59	1000	2484	2484.15	2484.17	2484.04
60	1000	2218	2218.01	2218.03	2218
61	1000	2450	2450	2450.02	2450
62	1000	2462	2462.03	2462.02	2462
63	1000	2550	2550.03	2550.2	2550.02
64	1000	2325	2325.19	2325.15	2325.1
65	1000	2472	2472.03	2472.01	2472.02
66	1000	2591	2591.35	2591.2	2591.2
67	1000	2463	2463.15	2463.21	2463.05
68	1000	2248	2248.07	2248.13	2248.05
69	1000	2441	2441.06	2441.02	2441.02
70	1000	2502	2502.22	2502.16	2502.11
71	1000	2638	2638.12	2638.21	2638.08
72	1000	2401	2401.19	2401.19	2401.1
73	1000	2449	2449.01	2449.04	2449
74	1000	2750	2750.23	2750.22	2750.14
75	1000	2514	2514.25	2514.2	2514.17
76	1000	2425	2425.11	2425.07	2425.03
77	1000	2723	2723.17	2723.21	2723.09
78	1000	2474	2474	2474.07	2474
79	1000	2224	2224.05	2224.11	2224.05
80	1000	2534	2534.01	2534.03	2534.02
81	1000	2656	2656.01	2656.03	2656.01
82	1000	2770	2770.04	2770.01	2770.05
83	1000	2420	2420.08	2420.11	2420.06

84	1000	2467	2467.03	2467.01	2467.03
85	1000	2695	2695	2695.05	2695
86	1000	2639	2639.02	2639.04	2639.02
87	1000	2678	2678.15	2678.22	2678.09
88	1000	2745	2745.21	2745.23	2745.09
89	1000	2718	2718.09	2718.14	2718.07
90	1000	2520	2520.34	2520.28	2520.18
91	1000	2539	2539.16	2539.26	2539.07
92	1000	2432	2432.05	2432.14	2432.01
93	1000	2434	2434.02	2434.02	2434
94	1000	2242	2242.01	2242.05	2242
95	1000	2618	2618.02	2618.09	2618
96	1000	2495	2495.03	2495.05	2495
97	1000	2645	2645.21	2645.19	2645.1
98	1000	2628	2628.02	2628.02	2628.01
99	1000	2301	2301.05	2301.01	2301

Figure A-7: Average Fitness Result(MaxIteration = 1000)

Instance	Max iteration	Best solution	Medium solution(SBM)	Medium solution(HM)	Medium solution(PHM)
0	1000	2691	2691	2691	2691
1	1000	2439	2439	2439	2439
2	1000	2532	2532	2532	2532
3	1000	2599	2599	2599	2599
4	1000	2639	2639	2639	2639
5	1000	2375	2375	2375	2375
6	1000	2571	2571	2571	2571
7	1000	2490	2490	2490	2490
8	1000	2581	2581	2581	2581
9	1000	2575	2575	2575	2575
10	1000	2298	2298	2298	2298
11	1000	2554	2554	2554	2554
12	1000	2535	2535	2535	2535
13	1000	2441	2441	2441	2441
14	1000	2540	2540	2540	2540
15	1000	2457	2457	2457	2457
16	1000	2374	2374	2374	2374
17	1000	2468	2468	2468	2468
18	1000	2308	2308	2308	2308
19	1000	2409	2409	2409	2409
20	1000	2560	2560	2560	2560
21	1000	2487	2487	2487	2487

22	1000	2587	2587	2587	2587
23	1000	2344	2344	2344	2344
24	1000	2572	2572	2572	2572
25	1000	2493	2493	2493	2493
26	1000	2418	2418	2418	2418
27	1000	2388	2388	2388	2388
28	1000	2441	2441	2441	2441
29	1000	2516	2516	2516	2516
30	1000	2582	2582	2582	2582
31	1000	2335	2335	2335	2335
32	1000	2591	2591	2591	2591
33	1000	2560	2560	2560	2560
34	1000	2308	2308	2308	2308
35	1000	2226	2226	2226	2226
36	1000	2499	2499	2499	2499
37	1000	2445	2445	2445	2445
38	1000	2502	2502	2502	2502
39	1000	2499	2499	2499	2499
40	1000	2514	2514	2514	2514
41	1000	2588	2588	2588	2588
42	1000	2493	2493	2493	2493
43	1000	2645	2645	2645	2645
44	1000	2682	2682	2682	2682
45	1000	2617	2617	2617	2617
46	1000	2594	2594	2594	2594
47	1000	2627	2627	2627	2627
48	1000	2440	2440	2440	2440
49	1000	2515	2515	2515	2515
50	1000	2670	2670	2670	2670
51	1000	2522	2522	2522	2522
52	1000	2527	2527	2527	2527
53	1000	2513	2513	2513	2513
54	1000	2611	2611	2611	2611
55	1000	2818	2818	2818	2818
56	1000	2635	2635	2635	2635
57	1000	2453	2453	2453	2453
58	1000	2724	2724	2724	2724
59	1000	2484	2484	2484	2484
60	1000	2218	2218	2218	2218
61	1000	2450	2450	2450	2450
62	1000	2462	2462	2462	2462
63	1000	2550	2550	2550	2550

64	1000	2325	2325	2325	2325
65	1000	2472	2472	2472	2472
66	1000	2591	2591	2591	2591
67	1000	2463	2463	2463	2463
68	1000	2248	2248	2248	2248
69	1000	2441	2441	2441	2441
70	1000	2502	2502	2502	2502
71	1000	2638	2638	2638	2638
72	1000	2401	2401	2401	2401
73	1000	2449	2449	2449	2449
74	1000	2750	2750	2750	2750
75	1000	2514	2514	2514	2514
76	1000	2425	2425	2425	2425
77	1000	2723	2723	2723	2723
78	1000	2474	2474	2474	2474
79	1000	2224	2224	2224	2224
80	1000	2534	2534	2534	2534
81	1000	2656	2656	2656	2656
82	1000	2770	2770	2770	2770
83	1000	2420	2420	2420	2420
84	1000	2467	2467	2467	2467
85	1000	2695	2695	2695	2695
86	1000	2639	2639	2639	2639
87	1000	2678	2678	2678	2678
88	1000	2745	2745	2745	2745
89	1000	2718	2718	2718	2718
90	1000	2520	2520	2520	2520
91	1000	2539	2539	2539	2539
92	1000	2432	2432	2432	2432
93	1000	2434	2434	2434	2434
94	1000	2242	2242	2242	2242
95	1000	2618	2618	2618	2618
96	1000	2495	2495	2495	2495
97	1000	2645	2645	2645	2645
98	1000	2628	2628	2628	2628
99	1000	2301	2301	2301	2301

Figure A-8: Medium Fitness Result(MaxIteration = 1000)

Instance	Max iteration	Best solution	Worst solution(SBM)	Worst solution(HM)	Worst solution(PHM)
0	1000	2691	2692	2693	2691

1	1000	2439	2440	2441	2440
2	1000	2532	2534	2534	2533
3	1000	2599	2600	2600	2600
4	1000	2639	2640	2640	2640
5	1000	2375	2376	2376	2375
6	1000	2571	2573	2572	2572
7	1000	2490	2491	2491	2491
8	1000	2581	2583	2582	2582
9	1000	2575	2576	2576	2575
10	1000	2298	2299	2300	2299
11	1000	2554	2556	2556	2555
12	1000	2535	2536	2536	2535
13	1000	2441	2443	2442	2442
14	1000	2540	2542	2542	2541
15	1000	2457	2458	2458	2457
16	1000	2374	2375	2375	2375
17	1000	2468	2469	2469	2469
18	1000	2308	2309	2308	2309
19	1000	2409	2410	2410	2410
20	1000	2560	2561	2561	2561
21	1000	2487	2488	2488	2488
22	1000	2587	2588	2589	2588
23	1000	2344	2345	2345	2345
24	1000	2572	2573	2573	2572
25	1000	2493	2494	2494	2494
26	1000	2418	2419	2419	2419
27	1000	2388	2389	2389	2389
28	1000	2441	2442	2442	2442
29	1000	2516	2517	2518	2517
30	1000	2582	2583	2583	2582
31	1000	2335	2336	2336	2335
32	1000	2591	2592	2592	2592
33	1000	2560	2562	2562	2561
34	1000	2308	2310	2309	2309
35	1000	2226	2227	2227	2227
36	1000	2499	2500	2500	2499
37	1000	2445	2445	2446	2445
38	1000	2502	2503	2503	2503
39	1000	2499	2500	2500	2500
40	1000	2514	2515	2515	2514
41	1000	2588	2589	2589	2588
42	1000	2493	2494	2494	2493

43	1000	2645	2646	2646	2646
44	1000	2682	2683	2684	2683
45	1000	2617	2618	2618	2617
46	1000	2594	2595	2595	2594
47	1000	2627	2629	2628	2628
48	1000	2440	2442	2442	2441
49	1000	2515	2517	2516	2515
50	1000	2670	2671	2671	2671
51	1000	2522	2524	2523	2523
52	1000	2527	2530	2528	2528
53	1000	2513	2514	2514	2514
54	1000	2611	2612	2613	2612
55	1000	2818	2819	2819	2819
56	1000	2635	2636	2636	2636
57	1000	2453	2454	2454	2453
58	1000	2724	2726	2725	2725
59	1000	2484	2485	2485	2485
60	1000	2218	2219	2219	2218
61	1000	2450	2450	2451	2450
62	1000	2462	2463	2463	2462
63	1000	2550	2551	2551	2551
64	1000	2325	2327	2326	2326
65	1000	2472	2473	2473	2473
66	1000	2591	2593	2592	2593
67	1000	2463	2464	2464	2464
68	1000	2248	2249	2249	2249
69	1000	2441	2442	2442	2442
70	1000	2502	2504	2503	2503
71	1000	2638	2640	2640	2639
72	1000	2401	2403	2403	2402
73	1000	2449	2450	2450	2449
74	1000	2750	2751	2751	2751
75	1000	2514	2515	2515	2515
76	1000	2425	2426	2426	2426
77	1000	2723	2725	2725	2724
78	1000	2474	2474	2476	2474
79	1000	2224	2225	2225	2225
80	1000	2534	2535	2536	2535
81	1000	2656	2657	2657	2657
82	1000	2770	2771	2771	2771
83	1000	2420	2421	2421	2421
84	1000	2467	2468	2468	2468

85	1000	2695	2695	2697	2695
86	1000	2639	2640	2640	2640
87	1000	2678	2679	2682	2679
88	1000	2745	2746	2747	2746
89	1000	2718	2719	2720	2719
90	1000	2520	2521	2522	2521
91	1000	2539	2541	2541	2540
92	1000	2432	2433	2434	2433
93	1000	2434	2435	2435	2434
94	1000	2242	2243	2243	2242
95	1000	2618	2619	2619	2618
96	1000	2495	2496	2496	2495
97	1000	2645	2647	2646	2647
98	1000	2628	2629	2629	2629
99	1000	2301	2302	2302	2301

Figure A-9: Worst Fitness Result(MaxIteration = 1000)

Instance	Max iteration	Best solution	Average solution(SBM)	Average solution(HM)	Average solution(PHM)
0	2000	2691	2691	2691	2691
1	2000	2439	2439.05	2439.02	2439
2	2000	2532	2532.04	2532.03	2532.02
3	2000	2599	2599	2599	2599
4	2000	2639	2639	2639	2639
5	2000	2375	2375	2375.01	2375
6	2000	2571	2571.02	2571.03	2571
7	2000	2490	2490	2490.03	2490
8	2000	2581	2581.1	2581.06	2581.03
9	2000	2575	2575	2575	2575
10	2000	2298	2298.06	2298.02	2298
11	2000	2554	2554.09	2554.02	2554.01
12	2000	2535	2535	2535	2535
13	2000	2441	2441.03	2441.01	2441.01
14	2000	2540	2540.05	2540.02	2540.02
15	2000	2457	2457.01	2457	2457
16	2000	2374	2374.03	2374.01	2374
17	2000	2468	2468	2468	2468
18	2000	2308	2308	2308	2308
19	2000	2409	2409	2409	2409
20	2000	2560	2560.01	2560	2560.01
21	2000	2487	2487.03	2487.03	2487.01
22	2000	2587	2587.07	2587.03	2587.01

23	2000	2344	2344.1	2344.05	2344.04
24	2000	2572	2572	2572	2572
25	2000	2493	2493	2493	2493
26	2000	2418	2418.02	2418.02	2418
27	2000	2388	2388.02	2388.02	2388
28	2000	2441	2441	2441	2441
29	2000	2516	2516.05	2516.01	2516
30	2000	2582	2582	2582	2582
31	2000	2335	2335	2335	2335
32	2000	2591	2591	2591	2591
33	2000	2560	2560.04	2560.02	2560
34	2000	2308	2308.05	2308.03	2308
35	2000	2226	2226.03	2226	2226
36	2000	2499	2499	2499	2499
37	2000	2445	2445	2445	2445
38	2000	2502	2502	2502	2502
39	2000	2499	2499	2499.04	2499
40	2000	2514	2514	2514	2514
41	2000	2588	2588	2588	2588
42	2000	2493	2493	2493.01	2493
43	2000	2645	2645	2645	2645
44	2000	2682	2682	2682	2682
45	2000	2617	2617	2617	2617
46	2000	2594	2594	2594	2594
47	2000	2627	2627.04	2627.04	2627.02
48	2000	2440	2440.1	2440.03	2440.03
49	2000	2515	2515	2515	2515
50	2000	2670	2670.08	2670.04	2670.03
51	2000	2522	2522.06	2522.04	2522.01
52	2000	2527	2527.01	2527	2527
53	2000	2513	2513	2513	2513
54	2000	2611	2611.02	2611.01	2611.02
55	2000	2818	2818.11	2818.03	2818.03
56	2000	2635	2635.05	2635.03	2635.03
57	2000	2453	2453	2453	2453
58	2000	2724	2724.11	2724.06	2724.04
59	2000	2484	2484	2484.03	2484.02
60	2000	2218	2218	2218	2218
61	2000	2450	2450	2450	2450
62	2000	2462	2462.01	2462	2462
63	2000	2550	2550	2550.03	2550
64	2000	2325	2325.07	2325.01	2325

65	2000	2472	2472	2472	2472
66	2000	2591	2591.08	2591.03	2591.03
67	2000	2463	2463.07	2463.03	2463.02
68	2000	2248	2248.05	2248.04	2248.01
69	2000	2441	2441.01	2441	2441
70	2000	2502	2502.08	2502.03	2502.02
71	2000	2638	2638.05	2638.04	2638.02
72	2000	2401	2401.03	2401.05	2401
73	2000	2449	2449	2449	2449
74	2000	2750	2750.08	2750	2750.01
75	2000	2514	2514.12	2514.06	2514.01
76	2000	2425	2425	2425	2425
77	2000	2723	2723.04	2723.02	2723.01
78	2000	2474	2474	2474	2474
79	2000	2224	2224.04	2224	2224
80	2000	2534	2534	2534	2534
81	2000	2656	2656	2656	2656
82	2000	2770	2770	2770	2770.01
83	2000	2420	2420.01	2420	2420.01
84	2000	2467	2467	2467	2467
85	2000	2695	2695	2695	2695
86	2000	2639	2639	2639	2639
87	2000	2678	2678.06	2678.06	2678.01
88	2000	2745	2745.05	2745.05	2745.03
89	2000	2718	2718.03	2718.05	2718
90	2000	2520	2520.09	2520.02	2520.01
91	2000	2539	2539.03	2539	2539.02
92	2000	2432	2432	2432.01	2432
93	2000	2434	2434	2434	2434
94	2000	2242	2242	2242	2242
95	2000	2618	2618	2618	2618
96	2000	2495	2495	2495.01	2495
97	2000	2645	2645.09	2645.05	2645.01
98	2000	2628	2628.01	2628.01	2628
99	2000	2301	2301	2301	2301

Figure A-10: Average Fitness Result(MaxIteration = 2000)

Instance	Max iteration	Best solution	Medium solution(SBM)	Medium solution(HM)	Medium solution(PHM)
0	2000	2691	2691	2691	2691
1	2000	2439	2439	2439	2439
2	2000	2532	2532	2532	2532

3	2000	2599	2599	2599	2599
4	2000	2639	2639	2639	2639
5	2000	2375	2375	2375	2375
6	2000	2571	2571	2571	2571
7	2000	2490	2490	2490	2490
8	2000	2581	2581	2581	2581
9	2000	2575	2575	2575	2575
10	2000	2298	2298	2298	2298
11	2000	2554	2554	2554	2554
12	2000	2535	2535	2535	2535
13	2000	2441	2441	2441	2441
14	2000	2540	2540	2540	2540
15	2000	2457	2457	2457	2457
16	2000	2374	2374	2374	2374
17	2000	2468	2468	2468	2468
18	2000	2308	2308	2308	2308
19	2000	2409	2409	2409	2409
20	2000	2560	2560	2560	2560
21	2000	2487	2487	2487	2487
22	2000	2587	2587	2587	2587
23	2000	2344	2344	2344	2344
24	2000	2572	2572	2572	2572
25	2000	2493	2493	2493	2493
26	2000	2418	2418	2418	2418
27	2000	2388	2388	2388	2388
28	2000	2441	2441	2441	2441
29	2000	2516	2516	2516	2516
30	2000	2582	2582	2582	2582
31	2000	2335	2335	2335	2335
32	2000	2591	2591	2591	2591
33	2000	2560	2560	2560	2560
34	2000	2308	2308	2308	2308
35	2000	2226	2226	2226	2226
36	2000	2499	2499	2499	2499
37	2000	2445	2445	2445	2445
38	2000	2502	2502	2502	2502
39	2000	2499	2499	2499	2499
40	2000	2514	2514	2514	2514
41	2000	2588	2588	2588	2588
42	2000	2493	2493	2493	2493
43	2000	2645	2645	2645	2645
44	2000	2682	2682	2682	2682

45	2000	2617	2617	2617	2617
46	2000	2594	2594	2594	2594
47	2000	2627	2627	2627	2627
48	2000	2440	2440	2440	2440
49	2000	2515	2515	2515	2515
50	2000	2670	2670	2670	2670
51	2000	2522	2522	2522	2522
52	2000	2527	2527	2527	2527
53	2000	2513	2513	2513	2513
54	2000	2611	2611	2611	2611
55	2000	2818	2818	2818	2818
56	2000	2635	2635	2635	2635
57	2000	2453	2453	2453	2453
58	2000	2724	2724	2724	2724
59	2000	2484	2484	2484	2484
60	2000	2218	2218	2218	2218
61	2000	2450	2450	2450	2450
62	2000	2462	2462	2462	2462
63	2000	2550	2550	2550	2550
64	2000	2325	2325	2325	2325
65	2000	2472	2472	2472	2472
66	2000	2591	2591	2591	2591
67	2000	2463	2463	2463	2463
68	2000	2248	2248	2248	2248
69	2000	2441	2441	2441	2441
70	2000	2502	2502	2502	2502
71	2000	2638	2638	2638	2638
72	2000	2401	2401	2401	2401
73	2000	2449	2449	2449	2449
74	2000	2750	2750	2750	2750
75	2000	2514	2514	2514	2514
76	2000	2425	2425	2425	2425
77	2000	2723	2723	2723	2723
78	2000	2474	2474	2474	2474
79	2000	2224	2224	2224	2224
80	2000	2534	2534	2534	2534
81	2000	2656	2656	2656	2656
82	2000	2770	2770	2770	2770
83	2000	2420	2420	2420	2420
84	2000	2467	2467	2467	2467
85	2000	2695	2695	2695	2695
86	2000	2639	2639	2639	2639

87	2000	2678	2678	2678	2678
88	2000	2745	2745	2745	2745
89	2000	2718	2718	2718	2718
90	2000	2520	2520	2520	2520
91	2000	2539	2539	2539	2539
92	2000	2432	2432	2432	2432
93	2000	2434	2434	2434	2434
94	2000	2242	2242	2242	2242
95	2000	2618	2618	2618	2618
96	2000	2495	2495	2495	2495
97	2000	2645	2645	2645	2645
98	2000	2628	2628	2628	2628
99	2000	2301	2301	2301	2301

Figure A-11: Medium Fitness Result(MaxIteration = 2000)

Instance	Max iteration	Best solution	Worst solution(SBM)	Worst solution(HM)	Worst solution(PHM)
0	2000	2691	2691	2691	2691
1	2000	2439	2441	2440	2439
2	2000	2532	2533	2533	2533
3	2000	2599	2599	2599	2599
4	2000	2639	2639	2639	2639
5	2000	2375	2375	2376	2375
6	2000	2571	2572	2572	2571
7	2000	2490	2490	2491	2490
8	2000	2581	2582	2582	2582
9	2000	2575	2575	2575	2575
10	2000	2298	2299	2299	2298
11	2000	2554	2555	2555	2555
12	2000	2535	2535	2535	2535
13	2000	2441	2442	2442	2442
14	2000	2540	2541	2541	2541
15	2000	2457	2458	2457	2457
16	2000	2374	2375	2375	2374
17	2000	2468	2468	2468	2468
18	2000	2308	2308	2308	2308
19	2000	2409	2409	2409	2409
20	2000	2560	2561	2560	2561
21	2000	2487	2488	2488	2488
22	2000	2587	2588	2588	2588
23	2000	2344	2345	2345	2345
24	2000	2572	2572	2572	2572

25	2000	2493	2493	2493	2493
26	2000	2418	2419	2419	2418
27	2000	2388	2389	2389	2388
28	2000	2441	2441	2441	2441
29	2000	2516	2517	2517	2516
30	2000	2582	2582	2582	2582
31	2000	2335	2335	2335	2335
32	2000	2591	2591	2591	2591
33	2000	2560	2561	2561	2560
34	2000	2308	2309	2309	2308
35	2000	2226	2227	2226	2226
36	2000	2499	2499	2499	2499
37	2000	2445	2445	2445	2445
38	2000	2502	2502	2502	2502
39	2000	2499	2499	2500	2499
40	2000	2514	2514	2514	2514
41	2000	2588	2588	2588	2588
42	2000	2493	2493	2494	2493
43	2000	2645	2645	2645	2645
44	2000	2682	2682	2682	2682
45	2000	2617	2617	2617	2617
46	2000	2594	2594	2594	2594
47	2000	2627	2628	2628	2628
48	2000	2440	2441	2441	2441
49	2000	2515	2515	2515	2515
50	2000	2670	2671	2671	2671
51	2000	2522	2523	2523	2523
52	2000	2527	2528	2527	2527
53	2000	2513	2513	2513	2513
54	2000	2611	2612	2612	2612
55	2000	2818	2819	2819	2819
56	2000	2635	2636	2636	2636
57	2000	2453	2453	2453	2453
58	2000	2724	2725	2725	2725
59	2000	2484	2484	2485	2485
60	2000	2218	2218	2218	2218
61	2000	2450	2450	2450	2450
62	2000	2462	2463	2462	2462
63	2000	2550	2550	2551	2550
64	2000	2325	2326	2326	2325
65	2000	2472	2472	2472	2472
66	2000	2591	2592	2592	2592

67	2000	2463	2464	2464	2464
68	2000	2248	2249	2249	2249
69	2000	2441	2442	2441	2441
70	2000	2502	2503	2503	2503
71	2000	2638	2640	2639	2639
72	2000	2401	2402	2402	2401
73	2000	2449	2449	2449	2449
74	2000	2750	2751	2750	2751
75	2000	2514	2515	2515	2515
76	2000	2425	2425	2425	2425
77	2000	2723	2724	2724	2724
78	2000	2474	2474	2474	2474
79	2000	2224	2225	2224	2224
80	2000	2534	2534	2534	2534
81	2000	2656	2656	2656	2656
82	2000	2770	2770	2770	2771
83	2000	2420	2421	2420	2421
84	2000	2467	2467	2467	2467
85	2000	2695	2695	2695	2695
86	2000	2639	2639	2639	2639
87	2000	2678	2679	2679	2679
88	2000	2745	2746	2746	2746
89	2000	2718	2719	2719	2718
90	2000	2520	2521	2521	2521
91	2000	2539	2540	2539	2540
92	2000	2432	2432	2433	2432
93	2000	2434	2434	2434	2434
94	2000	2242	2242	2242	2242
95	2000	2618	2618	2618	2618
96	2000	2495	2495	2496	2495
97	2000	2645	2646	2646	2646
98	2000	2628	2629	2629	2628
99	2000	2301	2301	2301	2301

Figure A-12: Worst Fitness Result(MaxIteration = 2000)