University of Sheffield

Randomized search heuristics for makespan scheduling

Wanchen Qu

Supervisor: Dr. Dirk Sudholt

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

in the

Department of Computer Science

September 12, 2018

Declaration

Name: Wanchen Qu

Signature:

Date: September 12, 2018

Abstract

Nowadays computers are used to solve complex problems. In the process of solving problems, the human brain cannot complete this task, and the exact algorithm takes a huge amount of time during the calculation process, so people need to develop an approximate algorithm known as heuristic algorithm. In that case heuristic algorithm may find an approximate solutions in acceptable time and space complexity. Evolutionary algorithm is one of the modern heuristic techniques.

The aim of this project is to analyse and evaluate the performance of various heuristic algorithms on solving the problem of makespan scheduling (or PARTITION), PARTITION means assigning several different sizes of jobs to two identical machines and keep the sum of the jobs on both machines as close as possible. The two major algorithm categories compared include Problem-Specific Heuristics for makespan scheduling, and General-Purpose Algorithms, such as the Evolutionary Algorithm and the Simulated Annealing Algorithm. So, another goal is to compare the performance of general-purpose and problem-specific algorithms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Modern issues are often very complex and involve large analytical data sets. Even if an exact algorithm can be developed, its time or space complexity may become unacceptable. However, in practice, it is usually sufficient to find an approximate or partial solution. Using this idea, we discuss some heuristic algorithms that solve the optimization problem approximately. The goal in such a problem is to find a solution that is as efficient as possible, which is to minimize or maximize the objective function[7]. The objective function is a function used to evaluate the quality of the generated solution. Many real-world problems can be called optimization problems. All possible solutions collected in a given problem can be considered as a search space, and the optimization algorithm is also the search algorithm in this search space.

Approximate algorithms raise interesting questions about the quality of the solution they find. Given that usually the best solution is unknown, this problem may be a real challenge involving powerful mathematical analysis.[9] In combination with quality issues, the goal of heuristics is to find the approximate optimal solution. There are some general heuristic strategies that can be successfully applied to many issues.

1.2 Aims and Objectives

The aim of this dissertation is identified as follows:

- Understand the concept of heuristic algorithms. Learn as much as possible about the principles and implementation of different heuristic algorithms.
- Design and implement general-purpose and problem-specific heuristics algorithms for makespan scheduling, in order to solve the problem of makespan scheduling on two identical machines.
- Differentiate the performance of various heuristics algorithms through various reasonable judgment conditions

1.3 Overview of the Report

The structure of this report, as well as the content of each chapter.

Chapter 1 (Introduction): This part provides the background information, and highlights the aims and objectives of this dissertation.

Chapter 2 (Literature Review): This chapter draws lessons from previous research topics. Details the background knowledge of makespan scheduling and heuristic algorithms, as well as the principle and execution process of the algorithm are described.

Chapter 3 (System Requirement and Analysis): This chapter will describe the functional and non-functional requirements for Java programming as Java will be used as the developing language. At the same time, the software project architecture will also be reflected in this chapter.

Chapter 4 (System Design and Implementation): The information covered in this section is as follows: System architectural design, code implementation of generic functions and code implementation of unique core ideas of each random search heuristic algorithms.

Chapter 5 (System Testing): This chapter will illustrate the data selection used for testing, and what the tester needs to do during the test.

Chapter 6 (Result and Analysis): Analyse and come to a conclusion based on the results obtained in the previous chapter.

Chapter 7 (Conclusions): This chapter looks back to the whole process of this project, and look into future work.

Chapter 2

Literature Review

In order to give readers a better understanding of the contents of the following chapters, this chapter will review some fundamental terminologies and conceptions of: Design Reasons for randomized search heuristics: NP Problem. The problem to be solved in this dissertation. The core idea of several randomized search heuristics algorithms.

2.1 P problem, NP problem, NP complete problem and NP hard problem

A problem in P is the easiest to solve and it can be done very quickly. Whereas a problem in NP cannot always be solved as quickly as a P problem, but it can promptly verify whether the answer is correct or not. While NP-hard problems are harder than all problems in NP but they are not necessarily in NP. Lastly an NP complete problem must be a problem in both NP and NP-hard.

Polynomial

$ax^n - bx^{n-1} + c$ is a polynomial of highest degree n

Time complexity

We know that when computing algorithm solves problems, we often use time and space complexity to represent the efficiency of an algorithm. Space complexity represents the amount of memory that an algorithm will occupy during the calculation process, whilst time complexity represents the computational workload required for this algorithm to run in order to obtain the desired solution. Time complexity discusses how quickly the algorithm requires changes in workload when the input value approaches infinity.

Sorting of time complexity: $O(1) < O(n) < O(n \lg n) < O(n^2) < O(n^a) < O(e^n)$ ($a > 2$, n represents the number of input data, $O(1)$ is a constant level)

2.1.1 P problem

According to Weisstein, Eric W.[20] P-problem is a problem where solution time is bounded by a polynomial. If a problem is known to be NP, and a solution is known, then demonstrating the correctness of the solution can always be reduced to a single P (polynomial time) verification. If P and NP are not equivalent, then the solution of NP-problems requires superpolynomial time. Superpolynomial time means algorithm running time is not bounded above by any polynomial.

We want to study whether there is a polynomial-time algorithm for a problem. We also only care about whether there is a polynomial algorithm for a problem, as any other algorithms that are more complex have no practical significance.

2.1.2 NP problem

The NP problem refers to the problem of verifying a solution in polynomial time. Another definition of the NP problem is the problem that you can guess a solution in polynomial time. We know that this problem is not an algorithm in polynomial time, so it is called non-deterministic, but we can verify and get a correct solution to this problem in polynomial time. A P problem is always in NP[5].

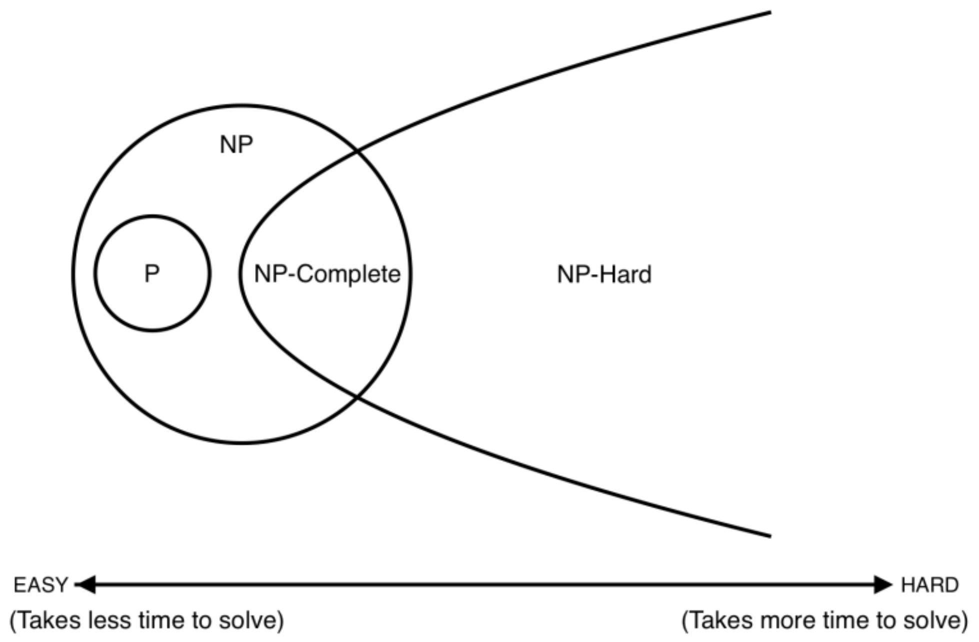


Figure 2.1: Connection between P, NP, NP-complete, NP-hard problem. Reproduced from https://williamswu.wordpress.com/2016/04/17/algorithm-np-p-nph-npc/ with the permission of the copyright owner.

2.1.3 NP complete problem

Reducibility

Reducibility means that problem A can be solved by the solution of problem B, or that problem A can be turned into problem B. An example is given in the Introduction to Algorithms[20]. For example, there are now two problems: solving a unary equation and solving a quadratic equation. Then we say that the former can be reduced to the latter, which means that if one knows how to solve a quadratic equation, then one can solve a one-dimensional equation. We can write two programs that correspond to two problems. Then we can find a 'rule' that makes the results of the two equations the same. This rule is: The coefficients of the corresponding terms of the two equations are unchanged, and the coefficient of the quadratic term of the quadratic equation is 0. According to this rule, the former question is converted into the latter question, and the two questions are equivalent.

NPC problem

The problem of satisfying the following two conditions at the same time is the NPC problem. First, it has to be an NP problem; then all NP problems can be reduced to it.

2.1.4 NP hard

The NPH problem satisfies the second condition defined by the NPC problem but does not necessarily satisfy the first condition. In other words, NP-Hard problem is wider than the NPC problem.

The problem to be solved in this article is PARTITION which is an NP-hard problem with an NP-complete decision variant. Hence, we cannot hope for exact solutions in polynomial time. However, according to Auger and Witt's research, randomized search heuristics are optimization algorithms that can be applied to a wide class of problems from combinatorial to continuous optimization.[1] So, the problem is perfectly suited for investigating the capabilities of stochastic search algorithms to approximate optimal solutions, which is also the main goal of this dissertation. And in the following sections, makespan scheduling and Partition problem are described in detail.

2.2 Makespan Scheduling

In this section, we consider the classical MAKESPAN SCHEDULING problem as defined in[12]. We are given m machines for scheduling, indexed by the set $M = \{1,...,m\}$. There are furthermore given n jobs, indexed by the set $J = \{1,...,n\}$, where job j takes $p_{i,j}$ units of time if scheduled on machine i. Let J_i be the set of jobs scheduled on machine i. Then $\ell_i = \sum_{j \in J_i} p_{i,j}$ is the makespan of the schedule.[12]

The problem is NP-hard, even if there are only two identical machines, so $p_{i,j} \to p_j$. However, we will derive several constant factor approximations and a *polynomial time* *approximation scheme(PTAS)* for identical machines and a 2-approximation for the general case. In this case, given n jobs with positive processing times $p_1, ..., p_n$, schedule them on two identical machines in a way such that the makespan, the overall completion time, is minimized. Let $x \in \{0,1\}^n$ be a decision vector. Job j is scheduled on machine 1 iff $x_j = 0$ holds and on machine 2 if and only if $x_j = i$ holds. Hence, the goal is to minimize[12]

$$f_{p_1,...,p_n}(x) := \max\left\{\sum_{i=1}^{n} p_j x_j, \sum_{i=1}^{n} p_j(1-x_j)\right\} \tag{2.1}$$

This project is trying to apply different randomized search heuristics in order to solve classic makespan scheduling problems in case of given two machines for scheduling. It means assigning several different sizes of jobs to two identical machines and keep the sum of the jobs on both machines as close as possible. We can see that this problem is easy to describe and leads to pseudo-boolean adaptation functions in a natural way. This is a very well-studied problem also known as **PARTITION**.

2.3 Partition problem

In number theory and computer science, the **partition problem**, or **number partitioning** is to determine whether a given set of positive integers S can be divided into two subsets S_1 and S_2 so that the sum of the numbers in S_1 is equal to the sum of the numbers in S_2.[2]

The completion time required for each job is different, and a certain number of jobs constitute a jobshop. Obviously, the size of different jobshops are uncertain, unified standard is required which needs to be chosen carefully. We need to minimise the difference in timSimulated Annealinge for two machines to finish all jobs assigned. However, the total completion time is uncertain, so only the completion time difference is incorrect. The standard (fitness function) used in this project is as follows:

$$fitness\,function: 1 - \left|\frac{sum\ of\ jobs\ assigned\ on\ machine\ A - sum\ of\ jobs\ assigned\ on\ machine\ B}{sum\ of\ all\ jobs}\right| \tag{2.2}$$

The closer the fitness value is to one, the better the job allocation is. So the goal of this project is to make value of fitness function as close as possible to one by applying different randomized search heuristics.

2.3.1 Pseudo-polynomial time algorithm

When the size of the set and the sum of integers in the set are not too large to make storage requirements infeasible, the problem can be solved using dynamic programming which is a method for solving a complex problem; by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions. The next time the same subproblem occurs, just simply look up the previously computed solution, thereby save computation time and space.[13]

The Pseudo-polynomial time algorithm can be described as follows: **Algorithm input**: A multiset S with size N, $S = \{x_1, ..., x_N\}$. Let K be the sum of all elements in S. $K = \sum_{i=1}^{n} x_i$. **Algorithm output**: whether there is a subset of S that sums to $\lfloor K/2 \rfloor$, if there is a subset, then: If K is even, the rest of S also sums to $\lfloor K/2 \rfloor$. If K is odd, then the rest of S sums to $\lfloor K/2 \rfloor$. This is as good a solution as possible.

Algorithm output is to determine whether there is a subset of S sums to $\lfloor K/2 \rfloor$. Let $p(i, j)$ be *True* if a subset of $\{x_1, ..., x_j\}$ sums to i and *False* otherwise. Then $p(\lfloor K/2 \rfloor, n)$ is *True* iff one of S's subset sums to $\lfloor K/2 \rfloor$. The recurrence relation are as follows: $p(i, j)$ is *True* if either $p(i, j-1)$ is *True* or if $p(1-x_j, j-1)$ is *True*. And $p(i, j)$ is *False* otherwise.[13]

pseudo-polynomial Pseudocode

Algorithm 1 The pseudo-polynomial algorithm[13]

```
1: $n \leftarrow |S|$
2: $K \leftarrow sum(S)$
3: $P \leftarrow$ empty boolean table of size $(\lfloor K/2 \rfloor + 1)$ by $(n + 1)$
4: initialize top row $P(0, x)$ of $P$ to True
5: initialize leftmost column $P(x, 0)$ of $P$ except for $P(0, 0)$ to False
6: for $i = 1$ to $\lfloor K/2 \rfloor$ do
7:     for $j = 1$ to $n$ do
8:         if $(i - S[j]) >= 0$ then
9:             $P(i, j) \leftarrow P(i, j-1)$ or $P(i - S[j], j) \leftarrow P(i, j-1)$
10:        else
11:            $P(i, j) \leftarrow P(i, j-1)$
12:            $result \leftarrow result + (middle - i)$
13:        end if
14:    end for
15: end for
16: return $P(\lfloor K/2 \rfloor, n)$
```

2.3.2 The Greedy Algorithm

Definition of Greedy Algorithm

The Greedy Algorithm means that when solving a problem, it always makes the best choice at present. In other words, do not consider the overall optimality, only make a local optimal solution. The Greedy Glgorithm is not an overall optimal solution for all problems. The key is the choice of greedy strategies. Greedy chosen strategy means the previous process of a certain state will not affect the future state, only the current state[11].

General steps for problem solving

- Establish a mathematical model to describe the problem
- Divide the problem of solving into several sub-problems
- Solve each subproblem to obtain the local optimal solution of the subproblem
- Combine the local optimal solution of the subproblem into a solution to the original problem

How Greedy Algorithm works in Partition problem

The sequence of digits is first sorted in descending order, and then each of the sequences is assigned to a subset with a smaller sum. This heuristic algorithm works well when the numbers in the collection are not much different, but it does not guarantee the best possible partition results.

This greedy approach is known to give a $\frac{7}{6}$ - *approximation* to the optimal solution of the optimization version. That is, if the Greedy Algorithm outputs two sets A and B, then $\max(\sum A, \sum B) \leq \frac{7}{6}OPT$, where OPT is the size of the larger set in the best possible partition.[12]

Greedy Algorithm Pseudocode

Algorithm 2 The Greedy Algorithm for partition problem

```
1: $A = set()$
2: $B = set()$
3: for $n$ in $int\_list$ do
4:     if $sum(A) < sum(B)$ then
5:         $A.add(n)$
6:     else
7:         $B.add(n)$
8:     end if
9: end for
10: return $(A, B)$
```

Insufficient of Greedy Algorithm

When dealing with worst case data. If the two largest jobs are assigned to the first and last respectively, the situation in Figure 2.2 will appear. Obviously, the performance of the algorithm is very poor in this case, and the solution is allocating large jobs first, which is the core idea of Longest procession time alogrithm.

Longest Procession Time

LPT is a good baseline algorithm using thought of Greedy Algorithm for the *PARTITION* problem. Greedy Algorithm is "go through all objects in the given order and put them on the emptier machine" without sorting them first. Whilst sorting them in a descending order would be LPT. So LPT IMHO is a special case of the Greedy Algorithm.

LPT Pseudocode

Algorithm 3 Longest Processing Timanalyzee(LPT)[12]

```
1: Sort the jobs according to decreasing processing time
2: for $i = 1, ..., n$ do
3:     Schedule the $ith$ job on the currently emptier machine(breaking ties arbitrarily)my
4: end for
```

As mentioned above, partition problem is perfectly suited for the investigation of the capabilities of stochastic search algorithms to approximate optimal solutions. In the following sections, this report will introduce some heuristic algorithms.

2.3.3 Existing work on EA for PARTITION

Representations and Neighborhood Structure

In a PARTITION problem, we are given n jobs and their processing time in nonincreasing order, i.e. $p_1 \geq ... \geq p_3$, $L \geq P/2$ represents possible barrier. And $P = \sum p_i$. And s^* represents *critical job size*, which means the smallest processing time of the jobs scheduled on the fuller machine.

Lemma 2.3.1. *[12] Let a current search point of RLS_b^1 or $(1+1)EA_b$ on an arbitary instance to the PARTITION problem be given. Suppose that the critical job size is guaranteed to be bounded from above by s' for all following search points of value greater than $L + s^*/2$. Then the algorithm reaches on f value at most $L + s^*/2$ in expected time $O(n^2)$*

And the following result is obtained by the method of expected multiplicative distance decrease.

Lemma 2.3.2. *[12] Let a current search point of RLS_b^1 or $(1+1)EA_b$ on an arbitary instance to the PARTITION problem be given. Suppose that the critical job size is guaranteed to be bounded from above by s' for all following search points of value greater than $L + s^*/2$. Then for any $\gamma > 1$ and $0 < \delta < 1$, $(1+1)EA_b$ (RLS_b^1) reaches an f value at most $L + s^*/2 + \delta P/2$ in at most $\lceil en \ln(\gamma/\delta) \rceil$ ($\lceil n \ln(\gamma/\delta) \rceil$) steps with probability at least $1 - \gamma^{-1}$. Moreover, the expected number of steps is at most $2\lceil en \ln(2/\delta) \rceil$ ($2\lceil n \ln(2/\delta) \rceil$)*

Worst-Case Analysis

Theorem 2.3.1. *[12] Let $\epsilon > 0$ be a constant. On every instance to the PARTITION problem, $(1+1)EA_b$ and RLS_b^1 reach an f value with approximation ratio $4/3+\epsilon$ in an expected number of $O(n)$ steps and an f value with approximation ratio $4/3$ in an expected number of $O(n^2)$ steps.*

It is almost tight for the algorithm to reach an approximate ratio of 4/3 in polynomial time. Here is a worst-case example called P_ϵ^*. In this case, both RLS and $(1+1)EA_b$ get stuck at approximation ratios close to 4/3 with probability $\Omega(1)$. Figure 2.2 is a worst-case instance, in this case there are many small jobs and two big jobs.

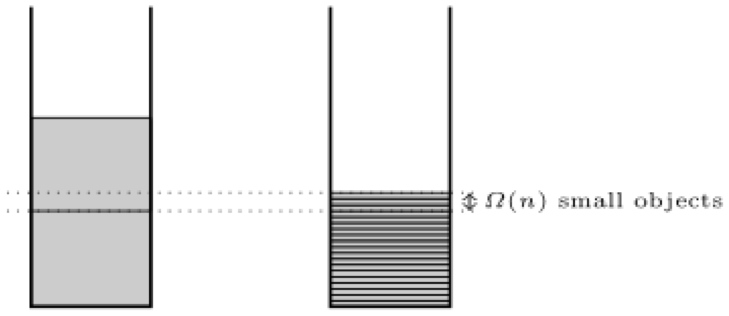


Figure 2.2: A worst-case instance[12]

Theorem 2.3.2. *[12] Let n be even and $\epsilon > 0$ be an arbitrarily small constant. Then the instance $P_\epsilon^* = \{p_1, ..., p_n\}$ is defined by $p_1 := p_2 := 1/3 - \epsilon/4$ and $p_i := (1/3+\epsilon/2)/(n-2)$ for $3 \leqslant i \leqslant n$*

In the following theorem, we only scheduled most of small jobs on the emptier machine.

Theorem 2.3.3. *[12] Let ϵ be any constant s.t. $0 < \epsilon < 1/3$. With probability $\Omega(1)$, both $(1+1)EA_b$ and RLS_b^1 need on the instance P_ϵ^* at least $n^{\Omega(1)}$ steps to create a solution with a better approximation ratio than $4/3-\epsilon$.*

Theorem 2.3.4. *[12] Let an arbitrary instance to the PARTITION problem be given and choose $\epsilon \geq 4/n$. Then, with probability at least $2^{-(e \log e)\lceil 2/\epsilon \rceil \ln(4/\epsilon) - \lceil 2/\epsilon \rceil}$, $(1+1)EA_b$ creates a solution of approximation ratio $(1+\epsilon)$ in $\lceil en \ln(4/\epsilon) \rceil$ steps. The same holds for RLS_b^1 with $\lceil n \ln(4/\epsilon) rceil$ steps and a probability of at least $2^{-(e \log e)\lceil 2/\epsilon \rceil \ln(4/\epsilon) - \lceil 2/\epsilon \rceil}$.*

Average-Case Analysis

Lemma 2.3.3. *[12] With probability $\Omega(1)$, the initial discrepancy of $(1+1)EA_b$ and RLS is $\Omega(\sqrt{n})$ in both the uniform and the exponential distribution models.*

Lemma 2.3.4. *[12] The discrepancy of $(1+1)EA_b$(RLS$_b^1$) in the uniform distribution model is bounded from above by 1 after an expected number of $O(n^2)$ *($O(n \log n)$) steps. Moreover, for any constant $c \geqslant 1$, it is bounded from above by 1 with probability at least $1 - O(1/n^c)$ after $O(n^2 \log n)$ ($O(n \log n)$) steps.*

2.4 Randomized Heuristic algorithm

The Randomized Heuristic algorithm has different definitions: one is defined as an algorithm based on intuitive or empirical construction, giving an acceptable calculation cost(calculation time, occupied space, etc.) for an instance of an optimization problem, giving an approximate optimal solution, the degree of deviation of the approximate solution from the true optimal solution may not be predictable. The other is that the heuristic algorithm is a technique that makes it possible to search for the best solution within the acceptable computational cost, but not guaranteed that is an acceptable solution or the optimal solution are obtained. Even in most cases, it is impossible to explain the approximate degree of the solution to the optimal solution. The popular explanation is to use similar principles of bionics to abstract certain phenomena in nature and animals into algorithms to deal with the corresponding problems. When a problem is an NP-hard problem, it is impossible to find the optimal solution. Therefore, a relatively good solution algorithm is used to approach the optimal solution as closely as possible. In many practical situations, tt is an acceptable technique.

2.4.1 Randomized local search

Randomized local search(RLS) can be seen as a simplification of the perhaps simplest Evolutionary Algorithm called (1+1)EA$_b$. In case of a runtime analysis of (1+1)EA$_b$, RLS is often considered as a first step and the results are later adjusted to the EA. Local search procedures: When run into the situation of there is no better solution in the neighbourhood than the current one. The algorithm stuck in local optima. Randomized local search (RLS) in the binary case produces from current solutions $s \in \{0,1\}^n$ a new one s' by flipping a randomly chosen bit[12].

RLS Pseudocode

Algorithm 4 RLS_b^1[12]

```
1: Choose $s \in \{0,1\}^n$ uniform at random.
2: Chose $s \in \{1,...,n\}$ uniform at random and flip the $i$th bit of $s$.
3: Replace $s$ with $s'$ if $f(s') < f(s)$.
4: Repeat Steps 2 and 3 forever.
```

RLS Problem

The main disadvantage of RLS is that it will fall into a local optimal solution, and may not be able to search for a global optimal solution. As shown in Figure 2.3 : Assume that point C is the current solution, and RLS stops searching after finding local optimal solution at point A, as looking at the diagram no matter which direction it moves to in a small range, a better solution cannot be obtained[4].

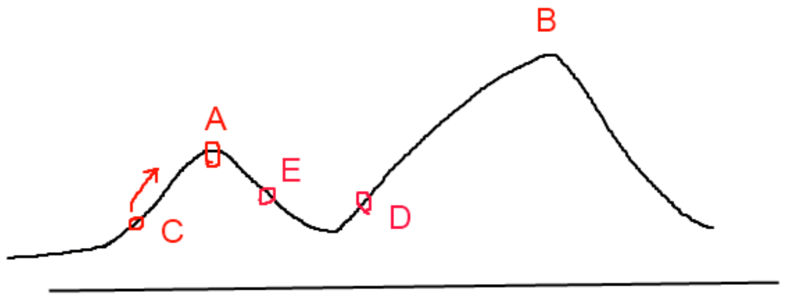


Figure 2.3: RLS fall into local optimum.

2.4.2 (1+1)EA$_b$

(1+1)EA$_b$ can be seen as variants of RLS with a more flexible mutation operator. The only difference between (1+1)EA$_b$ and RLS is, each search point of the considered search space should get a positive probability of being chosen in the next step.[12] In order to do this, producing a child at each iteration by flipping each bit in bit string with probability of $\frac{1}{n}$, n is the number of all jobs.

(1+1)EA$_b$ Pseudocode

Algorithm 5 (1+1)EA$_b$

```
1: Choose $s \in \{0,1\}^n$ uniform at random.
2: Produce $s'$ by flipping each bit of $s$ independently of the other bits with probability $\frac{1}{n}$
3: Replace $s$ with $s'$ if $f(s') < f(s)$.
4: Repeat Steps 2 and 3 forever.
```

2.4.3 Evolutionary Algorithm

According to Eibens research[3], the inspiration of the Evolutionary Algorithm draws on the evolutionary operations of nature. It generally includes basic operations such as gene coding, population initialization, crossover mutation operators, and operational retention mechanisms. Compared with traditional calculus-based methods and exhaustive methods, evolutionary computation is a mature global optimization method with high robustness and wide applicability. It has the characteristics of self-organization, self-adaptation and self-learning, meaning that EA can effectively deal with complex problems that traditional optimization algorithms find difficult to solve(such as NP hard problem).

EA as a flowchart[3]

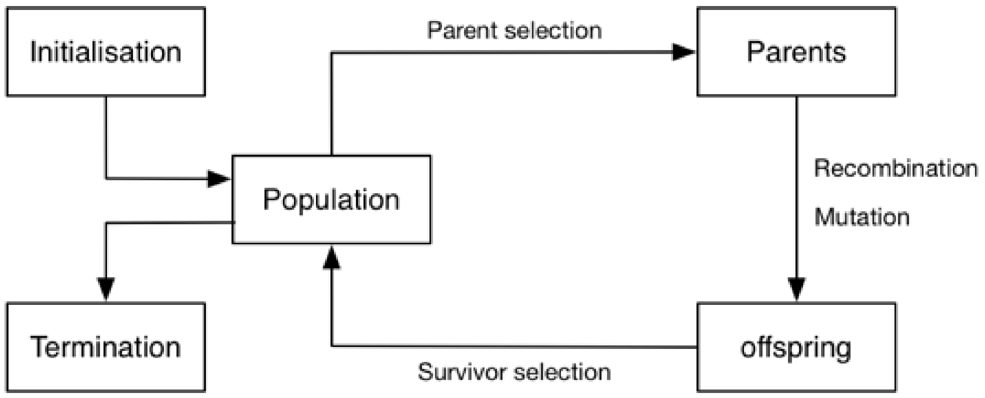


Figure 2.4: EA as a flowchart

EA Flowchart Explanation

Each Evolutionary Algorithm has four parameters: Population size, number of parent selection, recombination rate, and mutation rate.

Firstly, generate individuals with a population size add up to a population, for which the production of each individual is completely random. The next step is parent selection. During this process, EA chooses the number of parent selection individuals as parents and passes this to the next stage using "spin roulette wheel" strategy. Spin roulette wheel strategy means: Calculate the fitness value of each individual and put into roulette wheel. The higher the value, the higher the proportion in the roulette wheel, and the greater the probability that the roulette wheel will stop here after it is rotated. After getting offspring from the parents, the next step is rorganize and mutate. Add offspring into population for next generation. This is then looped until the termination condition is met.

Representation

Before the Implementation of Evolutionary Algorithm, the first step is representation, which is the definition of individuals, or a bridge link between the original problem and the EA problem. All possible solutions of the original problem, are known as phenotype space. And In the Evolutionary Algorithm, whether machine 1 or 2 processing ith job or not, it is represented by a gene(1/0), a combination of a set of genes is so called chromosome, genotype, or individual. Representation is the mapping from the phenotype to the genotype space.

Fitness assignment

Each problem will have a unique fitness function, for the makespan scheduling situation, this is:

$$f_{p_1,\dots,p_n}(x) := \max\left\{\sum_{i=1}^{n} p_j x_j, \sum_{i=1}^{n} p_j(1-x_j)\right\} \tag{2.3}$$

Parent selection mechanism

Distinguish among individuals based on their quality, which represent by fitness function. Allow better individuals to become parents for the next generation. But there is a small chance that low quality individuals will become parents of next generation, in order to make the whole search less greedy(otherwise the population will get stuck in a local optimum).

Survivor selection mechanism(Replacement)

Survivor Selection is often called the Replacement strategy: Generate a small number of children per iteration form a large number of population, this children is with low-quality and are removed for new ones.

Recombination and Mutation

Like chromosome crossover, recombination means randomly selecting the number of crossover points and then merges information from two parent genotypes into one or two offspring genotypes. And a mutation operator is always stochastic which output of recombination depends on the outcomes of a series of random choices. Stochastic variation in chromosomes increases the diversity in the population.

Termination Conditions

- There is not much change overall after doing x iterations.
- We have defined the number of evolutions in advance for the algorithm.
- Reaching the pre-defined fitness function value.
- The maximally allowed CPU time elapses.

EA Pseudocode

Algorithm 6 Evolutionary Algorithm[3]

```
1: INITIALIZE population with random candidate solutions;
2: EVALUATE each candidate;
3: while a termination criterion is not satisfied do
4:     SELECT parent
5:     RECOMBINE pairs of parents
6:     MUTATE the resulting offspring
7:     EVALUATE new candidates
8:     SELECT individuals for the next generation
9: end while
```

2.4.4 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an optimization algorithm established by simulation of swarm intelligence. Taking the example of random foraging of birds in a space, all birds do not know where foods are located, but they know how far they are. The easiest and most effective way is to search for the area around the bird that is currently closest to the food. So PSO sees birds as particles, and they have two properties: position and velocity. Then they change their flight direction based on the nearest solution to the food that they have found and refer to the most recent solution found throughout the entire cluster. Finally, we will find that the entire cluster is concentrated in the same place. This place is the nearest area to food, and food will be found when conditions are right.

Therefore, we need a *pbest* to record the optimal solution searched by individuals, and use *gbest* to record the optimal solution that the entire group searches in an iteration. The speed and particle position update formula is as follows:

$$v[i] = w * v[i] + c1 * rand() * (pbest[i] - present[i]) + c2 * rand() * (gbest - present[i]) \tag{2.4}$$

$$present[i] = present[i] + v[i] \tag{2.5}$$

PSO Pseudocode

Algorithm 7 basic PSO algorithm

```
1: Initialize the particle's position with a uniformly distributed random vector: $x_i \sim U(b_{1o}, b_{up})$
2: Initialize the particle's best known position to its initial position: $p_i \leftarrow x_i$
3: if $f(p_i) < f(g)$ then
4:     update the swarm's best known position: $g \leftarrow p_i$
5: end if
6: Initialize the particle's velocity: $v_i \sim U(-|b_{up} - b_{1o}|, |b_{up} - b1o|)$
7: while a termination criterion is not met do
8:     for each particle $i = 1, ..., S$ do
9:         for each dimension $d = 1, ..., n$ do
10:            Pick random numbers: $r_p, r_q \sim U(0, 1)$
11:            Update the particle's velocity: $v_{i,d} \leftarrow \omega v_{i,d} + \phi_p r_p(p_{i,d} - x_{i,d}) + \phi_g r_g(g_d - x_{i,d})$
12:        end for
13:        Update the particle's position: $x_i \leftarrow x_i + v_i$
14:        if $f(x_i) < f(p_i)$ then
15:            Update the particle's best known position: $p_i \leftarrow x_i$
16:            if $f(p_i) < f(g)$ then
17:                Update the swarm's best known position: $g \leftarrow p_i$
18:            end if
19:        end if
20:    end for
21: end while
```

2.4.5 Binary PSO

The original PSO was developed from solving continuous optimization problems. Eberhart et al. proposed a discrete binary version of PSO to solve the problem of combinatorial optimization in engineering practice. In the proposed model, each particle's history are optimally and globally optimally limited to 1 or 0, and the velocity is not limited to this. When updating the position with velocity, a threshold is set, and when the velocity is higher than the threshold, the position of the particle is taken as 1, otherwise it is taken as 0.[8]

In this binary space, position of one individual is abstracted into a bit string x_i. Position change in two-dimensional space with direction and distance, which is abstracted into the number of bits that change in the bit string. Velocity is abstracted into a velocity string v_n, each v_i represents the probability of bit x_i taking the value 1.

BPSO Pseudocode[16]

Algorithm 8 Binary PSO algorithm

```
1: velocity $v[n] \leftarrow 0^n$
2: Current best solution $x$ $and$ Golbal bese solution $x^{**} \leftarrow 0^n$
3: while a termination criterion is not met do
4:     for $i := 1$ to $n$ do
5:         set $x_i := 1$ with probability $s(v_i)$(as shown in Figure 2.5), otherwise set $x_i := 0$
6:         if $f(x) > f(x^{**})$ or $x^{**} = x$ then
7:             set $x^{**} = x$
8:         end if
9:         set $r$ = randomly chosen from 0 to 2
10:        for $i := 1$ to $n$ do
11:            set $v[i] := v + r * (x^{**}[i] - x[i])$
12:        end for
13:    end for
14: end while
```

Sigmoid function

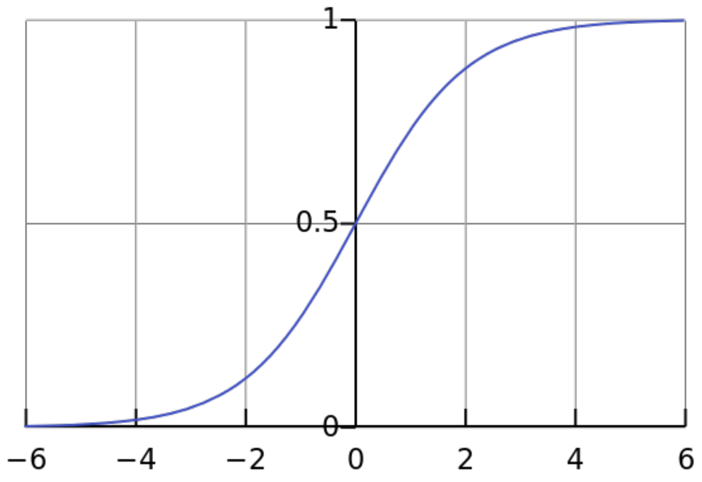


Figure 2.5: Sigmoid function. By Qef (talk) - Created from scratch with gnuplot, Public Domain, https://commons.wikimedia.org/w/index.php?curid=4310325.

2.4.6 Simulated Annealing Algorithm

Simple understanding of the Simulated Annealing Algorithm: In the process of a slow cooling to crystallisation after being heated to a higher temperature, similar to a thermodynamic system, the system energy tends to be the lowest as the temperature slowly decreases.

Core idea of Simulated Annealing Algorithm

The Simulated Annealing Algorithm contains both internal and external cycles. The external cycle is controlled by the temperature, and the temperature is determined by the initial temperature, and the temperature attenuation factor. The temperature is affected by the Metropolis criterion; the internal cycle is determined by the number of times it is set, and it mainly controls the number of new solutions generated at each temperature, which corresponds to the slow cooling process.[19]

In order to avoid falling into the local optimum, the Simulated Annealing Algorithm with a certain probability accepts a solution which is worse than the current solution, and the probability of accepting the error decreases as the number of iterations increases or as the temperature T decreases.

SA Pseudocode

Algorithm 9 Simulated Annealing Algorithm

```
1: $s \leftarrow GenerateInitialSolution()$
2: $T \leftarrow T_0$
3: cooling rate $c$
4: while a termination criterion is not met do
5:     $s \leftarrow PickAtRandom$
6:     neighbour solution $s' \leftarrow$ with chance of $\frac{1}{2}$, flip one bit a time, otherwise flip two bits
   a time
7:     if $f(s') >= f(s)$ then
8:         $s \leftarrow s'$
9:     else
10:        Accept worse solution $s'$ with probability $P(T, s', s)$
11:    end if
12:    $T \leftarrow T * (1 - c)$
13: end while
```

The algorithm starts with an initial solution that can be randomly selected or heuristically obtained, and initialize a temperature. During each iteration, a new solution is chosen at random from its neighbour. If the new solution has a better quality than the current one, then it is replaced. If the quality of the new solution is worse, then this new solution is accepted with a certain probability to replace the current one. Finally, update the temperature T for the next iteration. The probability of accepting a worse solution (It is worth mentioning that under these circumstances, $f(s') - f(s)$ is always negative) is a function of T and $f(s') - f(s)$. $M(s, s', T) = \min(1, e^{-\frac{f(s)-f(s')}{T}})$ where M is called the Metropolis function. The better the quality, the closer the value of $f(s)$ is to 1, and as shown in Figure 2.6, as T decreases, the less likely it is to accept a resolution.[12]

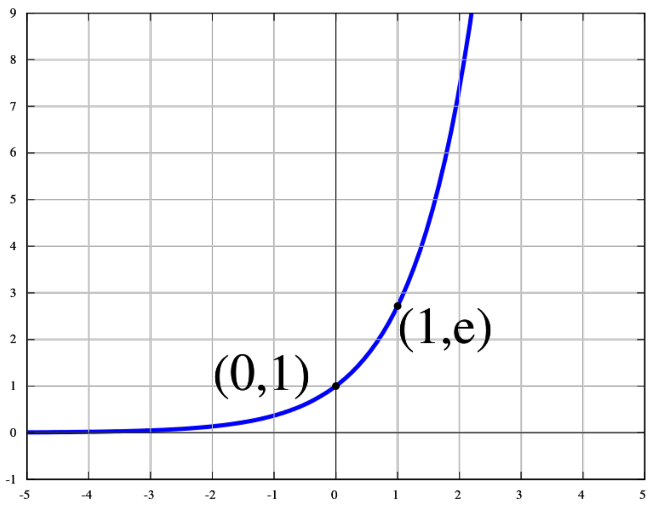


Figure 2.6: Exponential function. By Peter John Acklam - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1790833

2.4.7 Strong Selection Weak Mutation Regime(SSWM)

SSWM is similar with (1+1)EA, that both of them only maintains one genotype that may be replaced by mutated versions of it. The difference is that the SSWMs candidate solutions are accepted with probability:

$$p_{fix}(\Delta f) = \frac{1 - e^{-2\beta\Delta f}}{1 - e^{-2N\beta\Delta f}} \tag{2.6}$$

, where $\Delta f \neq 0$ is the fitness difference to the current solution, and $N \geq 1$ is the size of the underlying population. See figure 2.7 and extreme conditions see equation 2.6 [14]

$$\lim_{\beta\to\infty} p_{fix}(\Delta f) = \begin{cases} 0, if\Delta < 0 \\ 1/N, if\Delta = 0 \\ 1, if\Delta > 0 \end{cases} \tag{2.7}$$

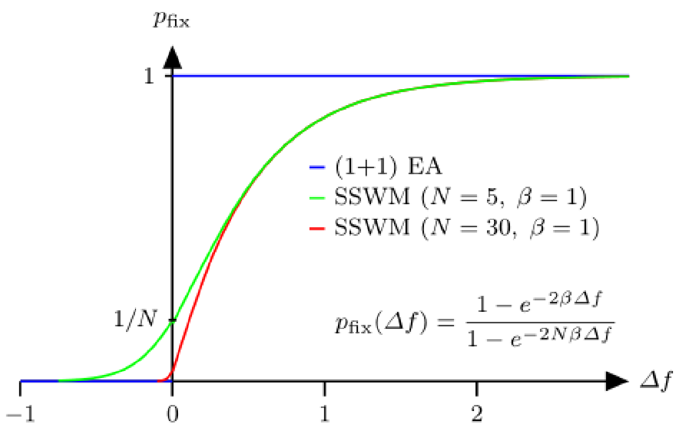


Figure 2.7: Probability of fixation of a new mutation that is initially present in one copy in the population[14]

SSWM Pseudocode

Algorithm 10 SSWM [14]

```
1: Choose $x \in \{0,1\}^n$ uniformly at random
2: while a termination criterion is not met do
3:     $y \leftarrow mutate(x)$
4:     $\Delta f = f(y) - f(x)$
5:     Choose $r \in [0,1]$ uniformly at random
6:     if $r < p_{fix}(\Delta f)$ then
7:         $x \leftarrow y$
8:     end if
9: end while
```

2.5 Summary

The essence of heuristics is a parallel, random, and direction-specific search method. This section summarises the basic elements, essence, and insufficiency of the heuristic algorithm.

2.5.1 Common features of heuristic algorithms

- Compared with the Gradient Descent method and similar methods, heuristics embody randomness.
- Compared to blind search algorithms, heuristics have certain directionality.

2.5.2 The basic elements of a heuristic algorithm

- Initialize a feasible solution randomly.
- Determination of evaluation function or objective function.
- The mechanism of the new solution.
- The acceptance mechanism of the new solution.
- Termination criteria.

2.5.3 Inadequacies of heuristic algorithms

- The heuristic algorithm currently lacks a unified and complete theoretical system.
- Due to the NP theory, various heuristic algorithms inevitably encounter local optimal problems.
- The parameters in the heuristic algorithm play an important role in the effect of the algorithm, and it is hard to set the parameters effectively.
- The heuristic algorithm lacks effective iteration stop conditions.
- Study of Heuristic algorithm convergence speed.

Chapter 3

System Requirement and Analysis

This section includes assessments of how to determine project objectives, deliverables, and evaluation plans, and how to determine these plans based on previous work. It also includes a subsection of ethical and legal issue[15].

3.1 Project Requirements

Requirements analysis is very important in the life cycle of a software project. The purpose of this dissertation is to study different random heuristic search algorithms and implement those algorithms in the form of coding. Run the program to solve the makespan scheduling problem, observe the performance of different algorithms. Requirements can be divided into two categories, functional requirements and non-functional requirements.

3.1.1 Introduction of functional and non-functional requirements

In general use, requirements are classified according to functional (behavioural) and non-functional (all other behaviours).The functional requirement is to say that there is a specific need to complete the content. Non-functional requirements refer to characteristics that software products must have in addition to functional requirements to meet user business needs, including system performance, reliability, maintainability, scalability, and adaptability to technology and business.

3.1.2 Functional Requirement

The space complexity and time complexity of the algorithm are collectively referred to as the complexity of the algorithm. How to choose from different algorithms mainly depends on these two aspects. The ideal situation is that the time and space complexity of an algorithm is small, but this is difficult to achieve. When facing different situations, we must analyse specific issues, whether it scarifies time for space or space for time. “Efficiency” of heuristic algorithm is difficult to define in a mathematical way. It can only be a guiding principle as follows.

Can effectively get a feasible solution

The heuristic algorithm should give corresponding parameters for special problems. The unreasonable choice of parameters will make the solution of a large probability not be within the feasible solution range, so that the search efficiency is extremely low. In view of this situation, we must carefully analyse the characteristics of the constraints, so that the new solution is roughly in line with the constraints. Only in this way can the efficiency of the algorithm be guaranteed.

Not easy to fall into local optimum situation

This is an important difference between heuristic and greedy algorithms. However, even if the algorithm is designed based on the principle of the original heuristic algorithm, it is still possible to fall into the local optimal solution. For example, in the simulated annealing algorithm, the probability acceptance function is always greater than 0, but if the value is very small, and the appropriate accuracy range is not set, the computer will display a result of 0, so that the result will no longer be accepted, which makes the simulated annealing algorithm fall into the dilemma of local optimal solution. This requires researchers to carefully collect data and optimize the initial and late data size or type differences.

Reasonably memorize existing data

It is certainly not feasible to store all the data that have been obtained. However, it seems unreasonable to only display the latest data, as it may have searched for an optimal solution during the calculation, and it will easily miss the optimal solution because it receives other solutions. Evolutionary algorithm can obtain multiple sets of solutions, it is reasonable to memorize the optimal solution in each generation. Corresponding data storage schemes are required for different algorithms, which can make the results more efficient.

3.1.3 Non-Functional Requirement

- Software system is able to handle incorrect user actions and input data.
- System response time should be reasonable.
- A developer who is familiar with the system should be able to quickly find the location of the parameter that needs to be modified by commenting the code. So that the system can be modified, observe and collect the needed experimental results.

3.2 Analysis

This section aims to break the problem down into manageable steps, which means that the developer and tester can obtain correct and reliable results.

3.2.1 Developer User Stories

The user story describe the functions that are valuable to the software (or system) user or customer. They are simply descriptions of requirements rather than detailed specifications. As a researcher. You can customize a jobshop by determining the total number of jobs and the time spent on each job. After getting jobshop, you can customize the parameters of the algorithm to observe the effect of different parameters on the performance of the algorithm. After determining the algorithmic process, you can set the number of algorithm runs and calculate the average to get a more representative result. After obtaining the experimental results, you can extract the data segments you are interested in for comparative analysis.

The developer user story of this system is presented as table 3.1:

3.2.2 Ethical, Professional and Legal Issues

According to the article "Ethical, Environmental and Legal Issues"[18]. When a computer system is designed and implemented it must meet legal requirements. Some of the laws that system designers should consider are as follows:

- The Data Protection Act says that anyone who stores personal details must keep them secure. Companies with computer systems that store any personal data must have processes and security mechanisms designed into the system to meet this requirement.
- The Health and Safety at Work Act makes employers responsible for their staff. Design considerations should provide appropriate working conditions for staff. Designers should consider how easy systems will be to use and any health implications there might be based on their choices of software, screen layout, input methods and the hardware used.
- The Copyright, Designs and Patents Act makes it illegal to use software without buying the appropriate licenses. When a computer system is designed and implemented licensing must be considered in terms of which software should be used. Is Open Source the way to go or is the cost of proprietary software worth it in the long run.[18]

According to the above characteristics, this dissertation does not violate these principles.

3.3 Testing and Evaluation

Selection of experimental data

Each set of experimental data is a jobshop. Every jobshop contains some different sizes of work. The size of each job means the time it takes to complete the job on the machine.

Test during development

In order to ensure the correctness of the experimental results, in each step of development, verify that the results of the current step are correct by printing.

No.	Actor	Action	Value of Action
1	As a researcher	Generate jobshop in *jobShopGenerator.java*	Change value of *job_size*, to set number of all jobs being assigned. Change method being used to generate different kinds of jobshop. Save into folder *jobShop*
2	As a researcher	Chose jobshop to test in *readFile.java*	Change the file name of jobshop being read
3	As a researcher	Run algorithms with no parameters	Including: *greedy algorithm*, *LPT algorithm*, *polynomial algorithm*,and $(1+1)EA$
4	As a researcher	Set algorithm parameters before running the genetic algorithm	EA contains four parameters: *population size*, *recombination rate*, *mutation rate*,and *number of parents selected.*
5	As a researcher	Set algorithm parameters before running the Simulated Annealing algorithm	SA contains two parameters: *system initial temperature*, and *system cooling rate*
6	As a researcher	Set algorithm parameters before running the Strong Selection Weak Mutation regime	SSWM contains two parameters: β, and Δf controlling the function that accept more poor solutions
7	As a researcher	Set the random number's upper limit	Usually set to 2.0, but still can be modified
8	As a researcher	After setting the algorithm parameters, program will run correctly. Researcher can determine the algorithm's run time to get an average running result.	30 and 100 are reasonable running time
9	As a researcher	Determine the required result interregional. Let the algorithm automatically run to the termination condition and get the results	Run the program and wait for the experimental results. Compare the performance of different algorithms to solve the same problem

Table 3.1: User Action Table

Chapter 4

System Design and Implementation

4.1 Systems development life cycle model

As the earliest software development model, the waterfall model provides the basic framework for software development and lays the foundation for the subsequent development model. It was widely adopted since it was proposed by Winston Royce in 1970 until the early 1980s.

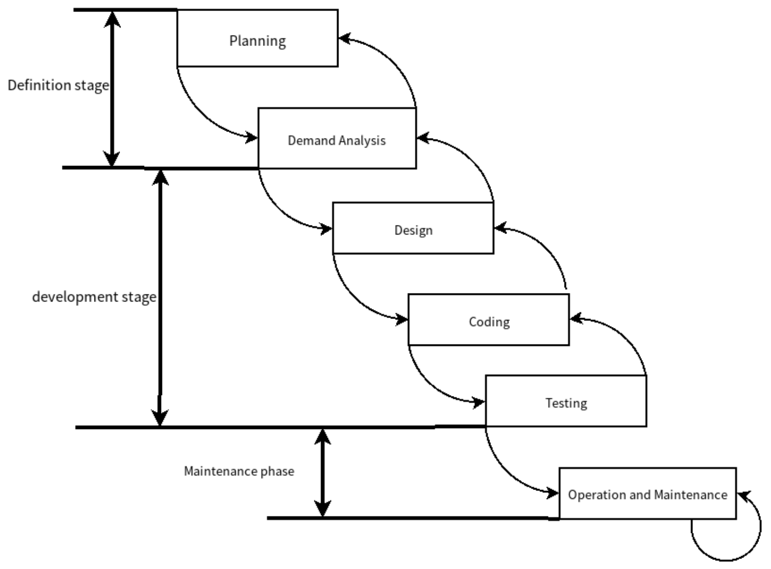


Figure 4.1: Waterfall model

This model has a clear phase of activities, the order of the phases is fixed, from top to bottom, connected to each other, shaped like a waterfall flowing down the ranks. This is also the origin of the name of the waterfall model. The essence of the waterfall model is one pass, that is, each activity is only executed once, and finally the software product is obtained, also called “linear order model” or “traditional life cycle”. The process of waterfall model is that the work of the previous stage completes the output and passes the review before it can flow to the next stage; otherwise, it returns to the previous stage[6].

One if the advantages of a waterfall model is that it provides a template for software development that allows analysis, design, coding, testing, and support methods to have a common guide under the template. It means checkpoints by stage are provided for the project, and after the current phase is completed, you only need to pay attention to the subsequent stages. This dissertation contains a large number of independent algorithms, and the relationship between algorithms is not very close. So it is reasonable to use the waterfall model for development.

4.2 Program framework design

There is no design pattern in this project. The program framework design means the general part except the core part of the algorithm: Data generation, Data acquisition, Data processing, etc.

4.2.1 Development language

Java is the programming language I am most familiar with, and there are a lot of choices in the library, whether it's core functionality or various extensions. Some core features can be replaced with many third-party libraries if you don't want to use the standard library. So I chose java as the project development language.

4.2.2 JobShop Design

Each set of experimental data is a jobshop. Every jobshop contains some different sizes of work. The size of each job means the time it takes to complete the job on the machine. Jobshop design are as follows:

All Ones

The simplest case: all jobs are the same size. Each size of word is the same, equivalent to only assign the number of jobs. If the size of jobshop is even, the number of jobs assigned on two machines is the same, the difference between the sum of jobs is 0. If the size of jobshop is odd, then the number of jobs assigned to the two machines differs by one, so the difference in the total work is 1.

Linear Changing

Job size equal to job index in jobshop.

Random Data

Randomly generate integer data within the specified range

Listing 4.1 Get Random Integer

```
/**
* get random Integer from min to max. Including min and max
*/
public static int getRandomNum(int min, int max) {
   Random random = new Random();
   int number = random.nextInt(max - min + 1) + 1;
   return number;
}
```

Worst Case

It is almost tight for the algorithm to reach an approximate ratio of 4/3 in polynomial time. Here is a worst-case example called P_{ϵ}^{*}. In this case, both RLS and $(1+1)EA_b$ get stuck at approximation ratios close to 4/3 with probability $\Omega(1)$. Figure 2.2 is a worst-case instance, in this case there are many small jobs and two big jobs.[12] This experiment selected 398 jobs of size 1 and 2 jobs of size 360 as jobshop.

Exponential function

Plan to use the exponential function to generate jobshop before the experiment. But because the value of the exponential function grows too fast as the number of jobs increases, there is not a variable format makes it easy to store data. So this dissertation did not take this jobshop formant.

4.2.3 Data Generation and Data Acquisition

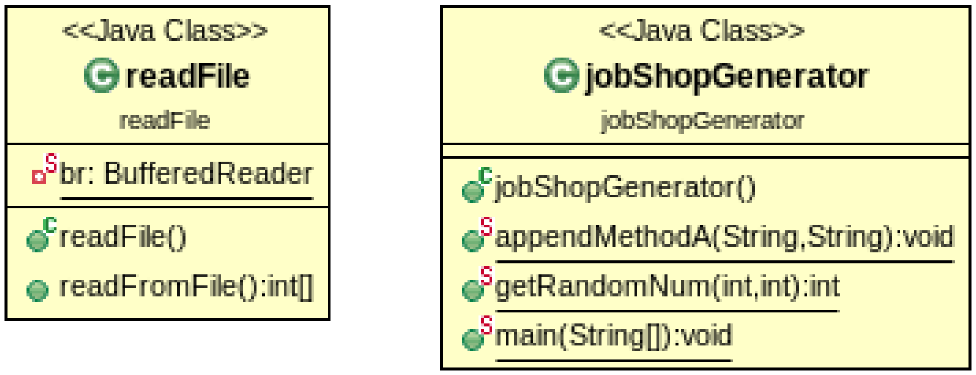


Figure 4.2: class diagram of *jobShopGenerator.java* and *readFile.java*

Read the contents of the selected file, return jobshop information as the form of integer array. And use it directly in the other function.

4.2.4 Data Processing

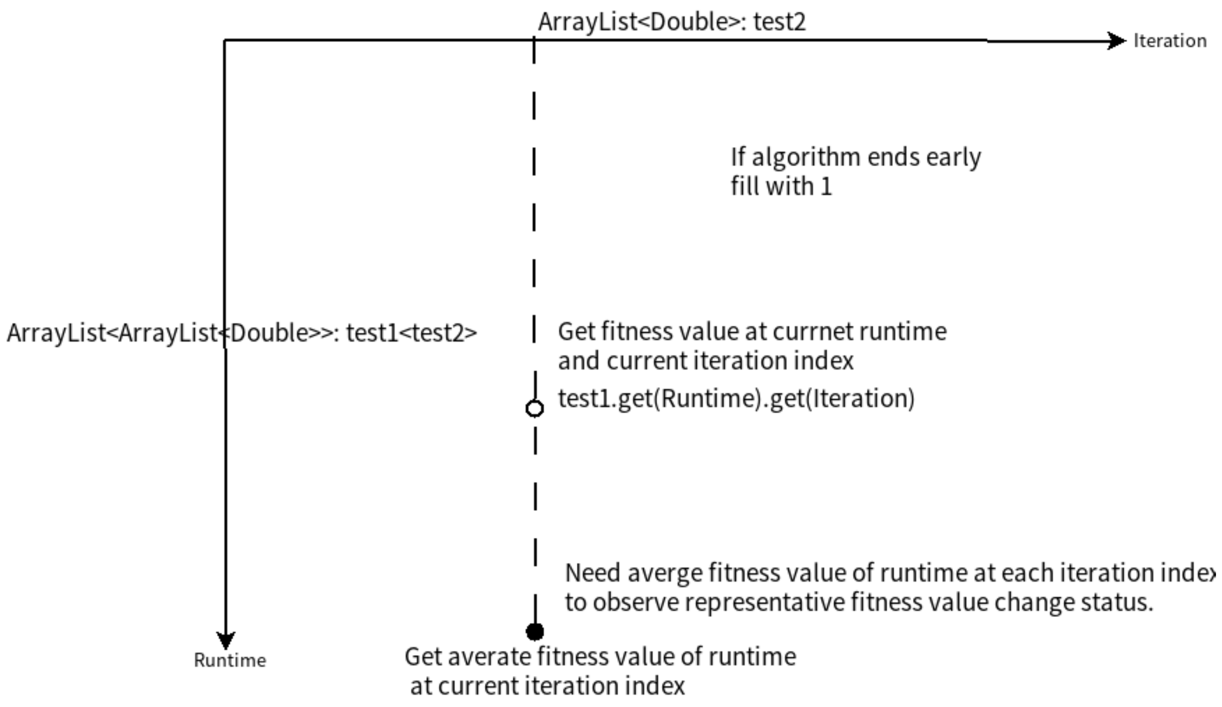

ArrayList<Double>: test2
Iteration
If algorithm ends early
fill with 1
ArrayList<ArrayList<Double>>: test1<test2>
Get fitness value at currnet runtime
and current iteration index
test1.get(Runtime).get(Iteration)
Need averge fitness value of runtime at each iteration inde
to observe representative fitness value change status.
Runtime
Get averate fitness value of runtime
at current iteration index

Figure 4.3: Data Processing

The number of iterations required by the randomized search heuristic algorithm to reach the termination condition is different for different runs. But the research needs an average fitness value of runtime at current iteration index, to get a representative data. But in this two-dimensional array, the data is obtained line by line, and what we need is the average of a column of data. So what I did is:

Listing 4.2 Data Processing

```
// run n times
int n = 1;
// init ArrayList<ArrayList<Double>> save Vertical axis info
// init ArrayList<Double> save Horizontal axis
ArrayList<ArrayList<Double>> test1 = new ArrayList<>();
ArrayList<Double> test2 = new ArrayList<>();
// for iteration for runtime count
for (int m = 0; m < n ;m++ ) {
   // clear arrayList test2 to let new infos in
   test2.clear();
   // iteration count
   int iterationCount = 1;
   double currentBestFitness;
   // while iteration for one time program end condition check
```

```
    while ("this iteration’s end condition not met") {
        // save current best fitness in a variable
        test2.add(currentBestFitness);
        iterationCount++;
    }
    // end while
    // copy fitness change process test2 into a new array testClone
    ArrayList<Double> testClone = (ArrayList<Double>) test2.clone();
    // add to test1
    test1.add(testClone);
}
```

After that, we will get a two-dimensional array with an indeterminate number of columns, which cannot take the average of every column. In order to see that the algorithm has ended early we need to set the results of the algorithm to 1, and to do this, we need to find the largest iteration in the running process, then fill empty part of result set with 1.0. Lastly, plot the results out in the form of chart.

4.3 Algorithmic Design

4.3.1 Individual Model

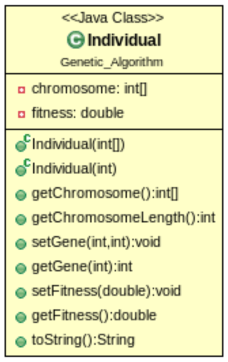


Figure 4.4: class diagram of *Individual.java*

In individual model, Defined two objects, chromosome and fitness. In addition to this, methods for operating the chromosome are defined.

4.3.2 Fitness Function

The essential content of all algorithms is a standard, a representative fitness function as we discussed in Section 2.3. Reflected in the code:

Listing 4.3 Fitness Function

```java
// calculate fitness for an individual
public static double calcFitness(Individual individual, int[] arr) {
  int arr_a[] = {};
  int arr_b[] = {};
  double difference;
  for (int geneIndex = 0; geneIndex < individual.getChromosomeLength(); geneIndex++) {
    if (individual.getGene(geneIndex) == 0) {
      arr_a = add(arr_a, arr[geneIndex]);
    } else if (individual.getGene(geneIndex) == 1) {
      arr_b = add(arr_b, arr[geneIndex]);
    }
  }
  difference = Math.abs(sumOfArr(arr_a) - sumOfArr(arr_b));
  double fitness1 = (double) (sumOfArr(arr) - difference) / sumOfArr(arr);
  // stroe fitness
  individual.setFitness(fitness1);
  return fitness1;
}
```

4.3.3 Greedy Algorithm and LPT Algorithm

Greedy Algorithm

The core idea of the greedy algorithm is to assign work to the current emptiest machine.

Listing 4.4 Greedy Algorithm

```java
// partition greedy
static void partition_greedy(int arr[], int arr_a[], int arr_b[]) {
  int n = arr.length;
  for (int i = 0; i < n; i++) {
    if (sumOfArr(arr_a) < sumOfArr(arr_b)) {
      arr_a = add(arr_a, arr[i]);
    } else {
      arr_b = add(arr_b, arr[i]);
    }
  }
}
```

LPT Algorithm

The core idea of LPT algorithm is to assign work according to greedy thoughts in descending order of work size. Sort jobshop using quick sort.

At first, there was no in-depth understanding of the meaning of the algorithm's order from large to small. I think it's just a matter of sorting, what I did was sort from the smallest to the largest and assign work. This gave very unreasonable results when testing the worst case. After encountering this problem, I understand that the purpose of sorting is to first allocate large jobs, and assign small jobs in order to make the overall difference as small as possible. So by sorting in a descending order resolved the problem and thus the algorithm is fixed.

4.3.4 RLS Algorithm and (1+1)EA

RLS Alogrithm

The core idea of RLS algorithm is to choose i in $1, ..., n$ uniform at random and flip the ith bit of s.

(1+1)EA

The core idea of (1+1)EA is to produce a child in each iteration by flipping each bit of s with prob $\frac{1}{n}$. Here is how to do it in Java:

Listing 4.5 (1+1)EA

```java
// produce in each iteration a child by flipping each bit of s with prob 1 / n
for (int chromosomeIndex = 0; chromosomeIndex < chromosomeLength; chromosomeIndex++) {
   if (Math.random() < flip_prob) {
      if (temp_individual.getGene(chromosomeIndex) == 1) {
         temp_individual.setGene(chromosomeIndex, 0);
      } else {
         temp_individual.setGene(chromosomeIndex, 1);
      }
   }
}
```

4.3.5 Evolutionary Algorithm

The core idea of evolutionary algorithm is to operate a population, which is a cluster of several individuals. Each population has 4 parameters, population number, crossover rate, mutation rate, and number of parent selected.

Roulette wheel strategy

Roulette wheel strategy is a selection method which uses an imaginary roulette wheel to select individuals from a population. Fitness value of each individual determines the proportion of each individual on the roulette. The higher the individual's fitness value, the higher the chance of selection. The meaning of roulette wheel select strategy is that all descendants have a chance to be selected, and the most fundamental purpose is to jump out of local optimum.

Listing 4.6 Rotating wheel strategy

```java
public Individual selectParent(Population population) {
  // get individuals
  Individual individuals[] = population.getIndividuals();
  // spin roulette whell
  double populationFitness = population.getPopulationFitness();
  double rouletteWheelPosition = Math.random() * populationFitness;
  // find parent
  double spinWhell = 0;
  for (Individual individual : individuals) {
    spinWhell += individual.getFitness();
    if (spinWhell >= rouletteWheelPosition) {
      return individual;
    }
  }
  return individuals[population.size() - 1];
}
```

Population Crossover and Mutation

Decide if it is needed to apply crossover and mutation to this individual. Crossover and mutation is only applied if both the crossover or mutation conditional is met, and the individual is not selected to join the next generation. Then perform crossover and mutation operations.

Listing 4.7 Crossover and Mutation

```java
// Crossover operation in function crossoverPopulation
for (int geneIndex = 0; geneIndex < parent1.getChromosomeLength(); geneIndex++) {
  // 50% parent1's gene, 50% parent2's gene
  if (0.5 > Math.random()) {
    offspring.setGene(geneIndex, parent1.getGene(geneIndex));
  } else {
    offspring.setGene(geneIndex, parent2.getGene(geneIndex));
  }
}
// Mutation operation in function mutatePopulation
if (this.mutationRate > Math.random()) {
  // get new gene
  int newGene = 1;
  if (individual.getGene(genIndex) == 1) {
    newGene = 0;
  }
  // mutate gene
  individual.setGene(genIndex, newGene);
}
```

We can modify the four parameters of the population during the test to observe interesting changes in results.

4.3.6 SA Algorithm and SSWM Algorithm

Both SA and SSWM are algorithms that are modified on the basis of (1+1)EA. They both add a probability function that accepts a more poor solution. In addition, SA defines a new way to generate new individuals.

SA Algorithm

The first time I designed the SA, I generated new individuals completely at random, and the algorithm was almost impossible to meet the requirements. So I revisited the reference and learned that the way SA generate new individuals is to “find a neighbour solution”, which I do not fully understand. Since my population was set to 400, I generated a neighbour solution by randomly flipping 5 bits. Although the correct solution was obtained, however the number of iterations was unacceptably large. This caused some confusion which was cleared by my supervisors advice.

The correct way to get the neighbour solution is to flip one bit at a time. However, in some cases it is impossible to get an optimal solution by flipping one point at a time. Figure 4.5 is an example.

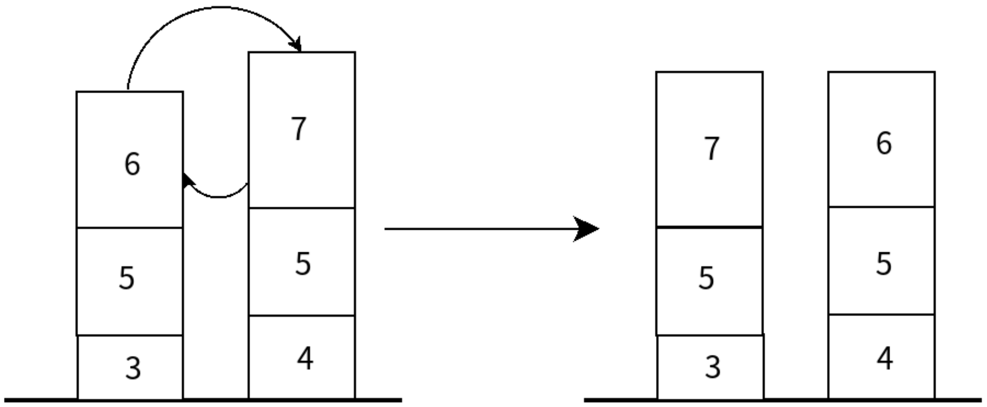


Figure 4.5: SA generate neighbour individual

So the most correct way is letting the probability of flipping one bit at a time and filping two bits a time be the same.

Listing 4.8 SA generate neighbour solution

```
// with chance of 0.5 flip 1 bit a time, otherwise flip 2 bits a time
if (Math.random() < 0.5) {
   for (int j = 0; j < 1; j++) {
      int random_index = getRandomNum(0, chromosomeLength - 2);
      reverseGene(neighbourIndividual, random_index);
   }
} else {
   for (int j = 0; j < 2; j++) {
      int random_index = getRandomNum(0, chromosomeLength - 2);
      reverseGene(neighbourIndividual, random_index);
   }
}
```

SA accept worse solution with probability $M(s, s', T) = \min(1, e^{-\frac{f(s)-f(s')}{T}})$ where M is the Metropolis function. The better the quality, the nearly the value of $f(s)$ to 1. And acroding to Figure 2.6, as T decreases, the less likely it is to accept a resolution.[12] System initial temperature controls initially probability to accept worse solution. System cooling rate controls the rate of decline in probability. For different problems we have to modify the system initial temperature and system cooling rate, in order to obtain a reasonable initial acceptance probability and a rate of decline in acceptance probability

Listing 4.9 SA accept worse solution probability

```
// calculate the accept probability
public static double acceptProb(double currentEnergy, double neighbourEnergy, double temperature)
   // neighbour energy is better than our current solution accept unconditionally
   if (neighbourEnergy > currentEnergy) {
      return 1.0;
   }
   // worse situation: accept function
   else {
      return Math.exp((neighbourEnergy - currentEnergy) / temperature);
   }
}
```

SSWM Algorithm

$$p_{fix}(\Delta f) = \frac{1 - e^{-2\beta\Delta f}}{1 - e^{-2N\beta\Delta f}} \tag{4.1}$$

The only difference between SSWM and (1+1)EA is adding the probability function of accrpting worse solution. Similar to SA, N controls initially accepts a worse solution probability, and β adjust Δf to a suitable interval. According to Figure 2.7 [-1, 3] is a suitable interval.

Listing 4.10 SSWM accept worse solution probability

```
// candidate solutions are accepted with probability
public static double calcProb(double delta_f, int beta, int N) {
   double accProb = (double) ((1 - Math.exp(-2 * beta * delta_f))
   / (1 - Math.exp(-2 * N * beta * delta_f)));
   return accProb;
}
```

4.3.7 Binary PSO Algorithm

The biggest problem encountered in designing the Binary PSO algorithm does not appear in the programming process. The problem was having troubles in understanding the transition from two-dimensional space to binary space, and the concept of individual and speed have changed in the process. Also misunderstood that a bit in a bit string is an individual. The correct understanding is: In binary space, the position of one individual is abstracted into a bit string x_i. Position change is number of bits that change in the bit string. And velocity is abstracted into a velocity string v_n, each v_i represents the probability of bit x_i taking the value 1.

The velocity change in binary space means change of position. Reverting to the previous model, this process generates a new way to assign jobshop. Here is the velocity update function.

Listing 4.11 Binary PSO sigmoid function

```
// sigmoid function
public static double[] sigmoid(double[] velocity) {
   int n = velocity.length;
   double[] s_velocity = new double[n];
   for (int i = 0; i < n; i++) {
      s_velocity[i] = 1 / (1 + Math.exp(-velocity[i]));
   }
   return s_velocity;
}
```

4.4 System class diagram design

The *Individual.java* and *readFile.java* are methods that are being used repeatedly by all algorithms. In *Individual.java*, a single chromosome is generated randomly, representing a makespan solution. In addition to this, there are several basic functions for manipulating chromosomes, and *readFile.java* reads the contents of the target file and store information as an array of integers for later convenience.

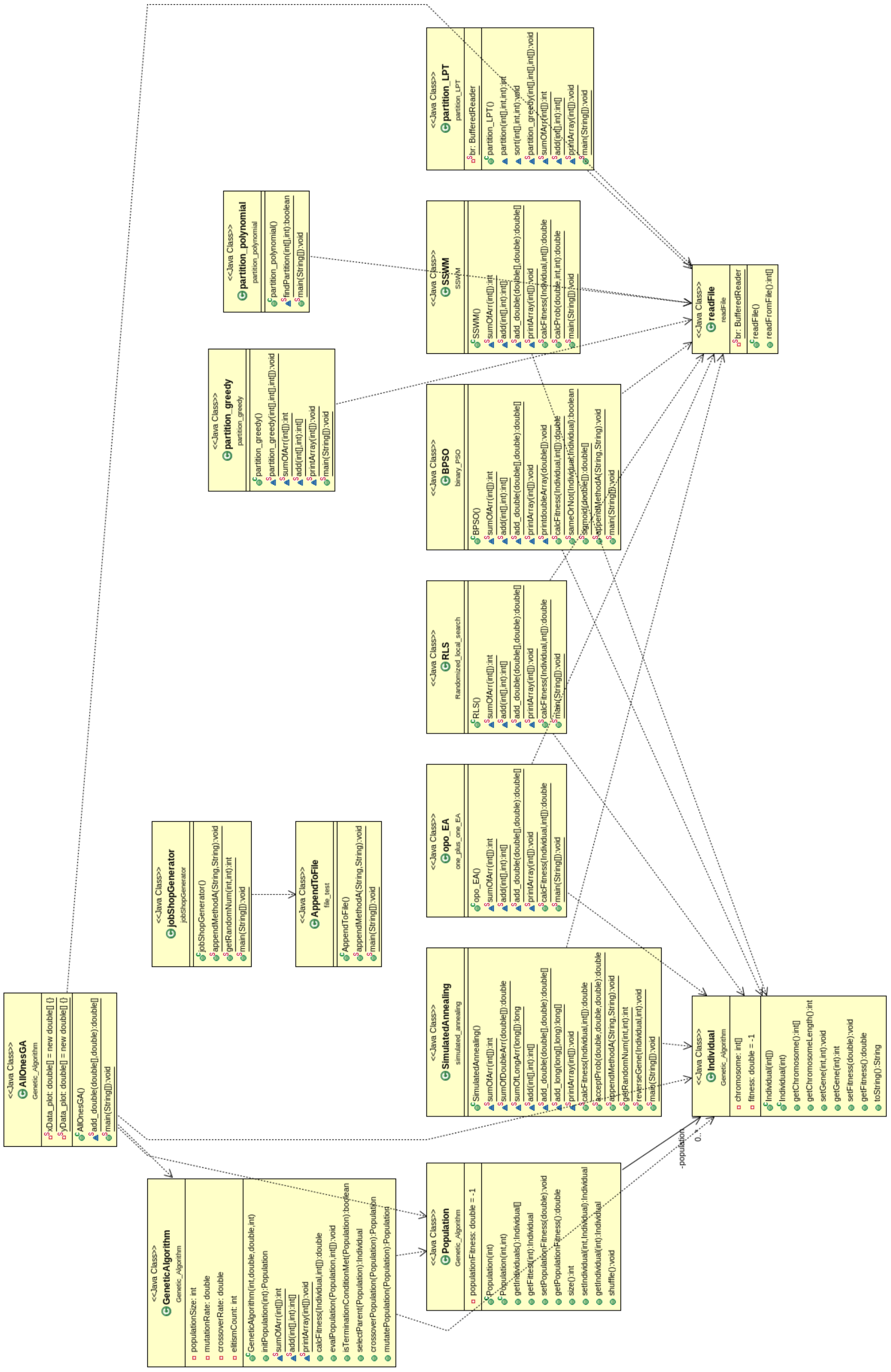


Figure 4.6: System class diagram design

All randomized search heuristics algorithm except evolutionary algorithm use *Individual.java* as a model for different operations. For evolutionary algorithm, the object of the algorithm operation is a population rather than a single individual, so the *Population.java* is needed. It is worth mentioning that in the *getFittes* method, the population is sorted in descending order with the individual fitness value. The first element of the sequence is the most powerful individual.

Listing 4.12 Get strongest individual in population

```java
// find an individual in the population by its fitness
// fitness: 0 is the strongest, population size - 1 is the weakest
// getFittest(0): strongest
public Individual getFittest(int offset) {
   // order population by fitness
   Arrays.sort(this.population, new Comparator<Individual>() {
      public int compare(Individual o1, Individual o2) {
         if (o1.getFitness() > o2.getFitness()) {
            return -1;
         } else if (o1.getFitness() < o2.getFitness()) {
            return 1;
         } else {
            return 0;
         }
      }
   });
   // return the fittest individual
   return this.population[offset];
}
```

Based on the *Population.java*, methods such as population assessment, roulette selection, crossover, mutation, and check termination conditions are in the *GeneticAlgorithm.java*.

4.5 Risk Identification and Quantification

Risk analysis is the review of risks associated with a particular event or action. It applies to projects, information technology, security issues and any actions that can analyse risk on a quantitative and qualitative basis. Risk analysis is an integral part of risk management.

The risk management process involves several key steps. First, identify potential threats. For example, the risks might be related to human error in using the computer, which creates a security risk. Risks are also related to projects that are not completed in a timely manner, resulting in significant costs. Next, quantitative and/or qualitative risk analysis is applied to study the identified risks. Quantitative risk analysis measures the predicted risk probability to predict the estimated financial loss of the potential risk[17]. Based on the above information, the risk assessment form is as follows.

Index	Risk	How risks will be managed	impact
1	Unclear the aim of final dissertation	Discuss with supervisor and identify the project expected result	3
2	Having difficulties in communicating with the supervisor during project design process	Get familiar with the pronunciation of some keywords, and always prepare before the project meeting	2
3	Changing requirements of the project	Identify the requirement with supervisor at the early stage	4
4	Unable to apply algorithms with Java programming language	Know how algorithms works at the early stage of this dissertation, and implement them with Java project	4
5	Unclear how to design algorithm performance evaluation section	Find relevant information first and discuss it with supervisor during project meetings	5
6	Problems about the essay structure	Get suggestions from supervisor, and draft an essay plan before starting it	2
7	Don't know how to make the poster	Search relevant information online	2
8	Illness/Emergency situations	Follow Sickness Absence Policy	1

Table 4.1: Risk Analysis Table

Chapter 5

System Testing

This chapter will illustrate the data selection used for testing. What the tester needs to do during the test, under the premise that the algorithm works correctly.

5.1 Test Data Selection

Size of Jobshop

Testers can modify the total number of jobs as needed. This number should not be too small, otherwise the algorithm will not give representative results. However, this number should not be too large either, as it will make the algorithm run for too long. For the above reasons, the total number of jobs in this experiment is chosen to be 400.

The Simplest Case

This experimental data consists of 400 jobs with job size of 1.

Most Complicated Situation

This experimental data consists of 398 jobs with job size of 1, and 2 jobs with job size of 360. In order to make the greedy algorithm get the worst solution, put the two biggest jobs at the beginning and end of the job sequence.

Completely Random Situation

The tester can modify the size interval of each job that is randomly generated.

Linear change

This experimental data consists of 400 jobs with job size of i, $i\ is\ from$ 1 to 400.

5.2 Parameters that can be modified

There are four algorithms that can modify parameters and observe changes in the results during the test. Evolutionary Algorithm, Simulated Annealing Algorithm, SSWM algorithm, and Binary PSO algorithm.

5.2.1 Evolutionary Algorithm Parameters

Four parameters in evolutionary algorithm, *populationSize*, *mutationRate*, *crossoverRate*, and *elitismCount*. These four parameters are used to describe population, which is the research object of evolutionary algorithm.

- *populationSize*: Amount of individuals that are chosen to constitute. The bigger the population size, the fewer generations it takes to find the optimal solution. But it doesn't mean speeding up the search, because the larger the *populationSize*, the longer it takes to check one generation.
- *elitismCount*: Elitism is the strongest member of the population which should be preserved from generation to generation. Cannot be larger than the *populationSize*, should be set to a relatively small value.
- *mutationRate* and *crossoverRate*: A value from 0 to 1. Representing the probability of parents' mutate and parents' crossover.

5.2.2 Simulated Annealing Algorithm Parameters

Two parameters in simulated annealing algorithm: System initial temperature *init_temp*, and system cooling rate *coolingRate*. Acroding to Figure 2.6, as T decreases, the less likely it is to accept a resolution.

We need to set a suitable *init_temp* according to the problem to get a reasonable initial acceptance probability. At the same time, it is necessary to set a suitable cooling rate to control the rate of decline of the acceptance probability.

5.2.3 SSWM Algorithm Parameters

Two parameters in SSWM algorithm: N and β. Similar to SA, N and β manage the changes in SSWM acceptance function. N controls initial acceptance probability, and β adjust Δf to a suitable interval, which is [-1, 3].

5.2.4 Binary PSO Algorithm Parameters

The only variable that can be modified here is $C1$. $C1$ is the upper limit of the random number in the speed update formula. Usually set to 2 based on pass experiences, so set $C1$ to 2 in the following test.

5.3 Test Case

According to the previous analysis, the test case is as shown in the table below.

No.	Test content	Test procedure	Expected test results
1	If new jobshop is needed, you can generate your own jobshop.	Modify the value of *job_size* in *jobShopGenerator.java* Modify the random number generation interval. Modify jobshop generation rules	Get the newly generated jobshop in the folder *jobShop*
2	Select the jobshop and algorithm parameters.	Select the jobshop used and determine the appropriate variables based on the jobshop selection.	Choose the appropriate variable values to embody the algorithm's characteristics or improve the performance of the algorithm
3	Set the number of algorithm runs	Determine the appropriate number of algorithm runs n to get representative results. Usually set n to 30 or 100	Algorithms run n times correctly
4	Collect experimental result for analysis	Save the resulting image file. You can also analyse the algorithm characteristics of a specific interval.	Obtain the resulting image and perform the subsequent analysis steps.

Table 5.1: Test Case Table

Chapter 6

Result and Analysis

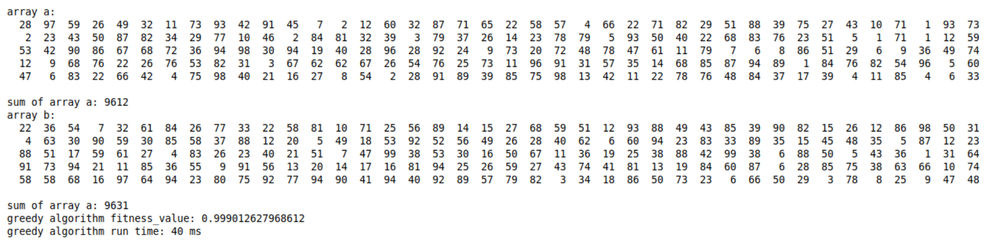
In this chapter, the algorithm performance tests are performed on the four experimental jobshops selected. The test criteria include the time and number of iterations to reach the algorithm termination condition. For the algorithms with parameters, use different combinations of parameters to conduct experiments, compare experimental results, and observe the influence of changes in parameters on the algorithm results. The large version of the experimental results figures in this chapter are in the appendix.

6.1 Randomly generated jobshop

Because the randomly generated jobshop generates work that is completely random and evenly distributed, it can be used as the most representative data to analyse algorithm performance.

6.1.1 Non-randomized search heuristics

Greedy Algorithm

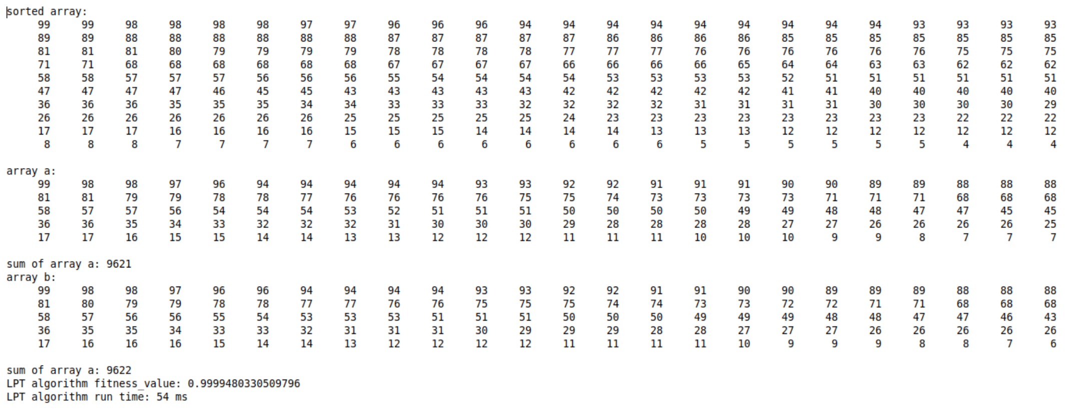


```
array a:
  28  97  59  26  49  32  11  73  93  42  91  45   7   2  12  60  32  87  71  65  22  58  57   4  66  22  71  82  29  51  88  39  75  27  43  10  71   1  93  73
   2  23  43  50  87  82  34  29  77  10  46   2  84  81  32  39   3  79  37  26  14  23  78  79   5  93  50  40  22  68  83  76  23  51   5   1  71   1  12  59
  53  42  90  86  67  68  72  36  94  98  30  94  19  40  28  96  28  92  24   9  73  20  72  48  78  47  61  11  79   7   6   8  86  51  29   6   9  36  49  74
  12   9  68  76  22  26  76  53  82  31   3  67  62  62  67  26  54  76  25  73  11  96  91  31  57  35  14  68  85  87  94  89   1  84  76  82  54  96   5  60
  47   6  83  22  66  42   4  75  98  40  21  16  27   8  54   2  28  91  89  39  85  75  98  13  42  11  22  78  76  48  84  37  17  39   4  11  85   4   6  33

sum of array a: 9612
array b:
  22  36  54   7  32  61  84  26  77  33  22  58  81  10  71  25  56  89  14  15  27  68  59  51  12  93  88  49  43  85  39  90  82  15  26  12  86  98  50  31
   4  63  30  90  59  30  85  58  37  88  12  20   5  49  18  53  92  52  56  49  26  28  40  62   6  60  94  23  83  33  89  35  15  45  48  35   5  87  12  23
  88  51  17  59  61  27   4  83  26  23  40  21  51   7  47  99  38  53  30  16  50  67  11  36  19  25  38  88  42  99  38   6  88  50   5  43  36   1  31  64
  91  73  94  21  11  85  36  55   9  91  56  13  20  14  17  16  81  94  25  26  59  27  43  74  41  81  13  19  84  60  87   6  28  85  75  38  63  66  10  74
  58  58  68  16  97  64  94  23  80  75  92  77  94  90  41  94  40  92  89  57  79  82   3  34  18  86  50  73  23   6  66  50  29   3  78   8  25   9  47  48

sum of array a: 9631
greedy algorithm fitness_value: 0.999012627968612
greedy algorithm run time: 40 ms
```

Figure 6.1: Greedy algorithm randomly generated jobshop test result

LPT Algorithm

According to Figure 6.1 and Figure 6.2, it can be seen that in solving the randomly generated jobshop makespan scheduling problem. The greedy algorithm get fitness value of 0.999012627968612



```
sorted array:
     99     99     98     98     98     98     97     97     96     96     96     94     94     94     94     94     94     94     94     94     93     93     93     93
     89     89     88     88     88     88     88     88     87     87     87     87     87     86     86     86     86     85     85     85     85     85     85     85
     81     81     81     80     79     79     79     79     78     78     78     78     77     77     77     76     76     76     76     76     76     75     75     75
     71     71     68     68     68     68     68     68     67     67     67     67     66     66     66     66     65     64     64     63     63     62     62     62
     58     58     57     57     57     56     56     56     55     54     54     54     54     53     53     53     53     52     51     51     51     51     51     51
     47     47     47     47     46     45     45     43     43     43     43     43     42     42     42     42     42     41     41     40     40     40     40     40
     36     36     36     35     35     35     34     34     33     33     33     32     32     32     32     31     31     31     31     30     30     30     30     29
     26     26     26     26     26     26     26     25     25     25     25     25     24     23     23     23     23     23     23     23     23     22     22     22
     17     17     17     16     16     16     16     15     15     15     14     14     14     14     13     13     13     12     12     12     12     12     12     12
      8      8      8      7      7      7      7      6      6      6      6      6      6      6      6      5      5      5      5      5      5      4      4      4

array a:
     99     98     98     97     96     94     94     94     94     94     93     93     92     92     91     91     91     90     90     89     89     88     88     88
     81     81     79     79     78     78     77     76     76     76     76     75     75     74     73     73     73     73     71     71     71     68     68     68
     58     57     57     56     54     54     54     53     52     51     51     51     50     50     50     50     49     49     48     48     47     47     45     45
     36     36     35     34     33     32     32     32     31     30     30     30     29     28     28     28     28     27     27     26     26     26     26     25
     17     17     16     15     15     14     14     13     13     12     12     12     11     11     11     10     10     10      9      9      8      7      7      7

sum of array a: 9621
array b:
     99     98     98     97     96     96     94     94     94     94     93     93     92     92     91     91     90     90     89     89     89     88     88     88
     81     80     79     79     78     78     77     77     76     76     75     75     75     74     74     73     73     72     72     71     71     68     68     68
     58     57     56     56     55     54     53     53     53     51     51     51     50     50     50     49     49     49     48     48     47     47     46     43
     36     35     35     34     33     33     32     31     31     31     30     29     29     29     28     28     27     27     27     26     26     26     26     26
     17     16     16     16     15     14     14     13     12     12     12     12     11     11     11     11     10      9      9      9      8      8      7      6

sum of array a: 9622
LPT algorithm fitness_value: 0.9999480330509796
LPT algorithm run time: 54 ms
```

Figure 6.2: LPT algorithm randomly generated jobshop test result

in 40 milliseconds, and the LPT algorithm takes 54 milliseconds to reach 0.9994403030509796.

Because LPT algorithm arrange the jobshop list in descending order on the basis of greedy algorithm, it takes longer and gets more reasonable results in the process of solving the problem. Moreover, LPT does not require the order of the initial sequence, but the change of the initial sequence order will influence the result of greedy algorithm.

6.1.2 RLS, (1+1)EA, BPSO

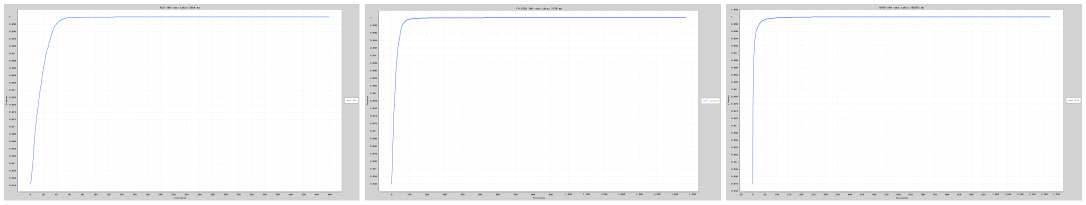


Figure 6.3: RLS, (1+1)EA, BPSO random jobshop

The overall experimental results of these three algorithms without parameters are shown as follows. Since the result of the change after 150 iterations is too small, select 2 to 150 iterations' data to analysis.

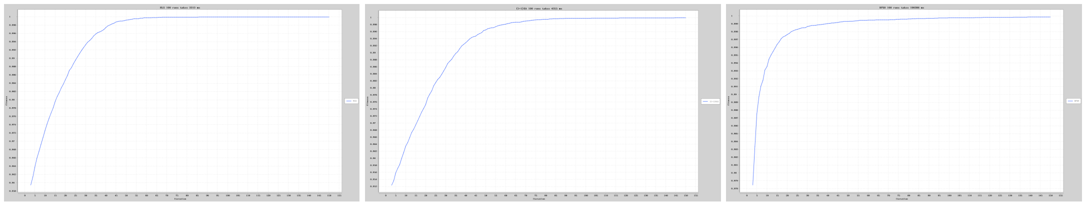


Figure 6.4: RLS, (1+1)EA, BPSO random jobshop cut from 2-150 iteration

Algorithm	Time Taken (millisecond)	Average fitness at 150 iterations	Largest Iteration	End fitness
RLS	3313	0.9999527100763912	459	0.9999994803305099
(1+1)EA	5578	0.9999012627968608	1661	0.9999994803305097
BPSO	192055	0.9999043808138021	1223	0.9999994803305099

Table 6.1: RLS, (1+1)EA, BPSO random jobshop result

It is shown by practice that RLS will give better experimental results in the shortest time, but there will be no solution if it falls into local optimum. BPSO can get better results with a minimum number of iterations, however it does take a lot of time. (1+1) EA does not fall into local optimum, but also solves problems within reasonable time and number of iterations.

6.1.3 Evolutionary Algorithm

Effect of population size on experimental results

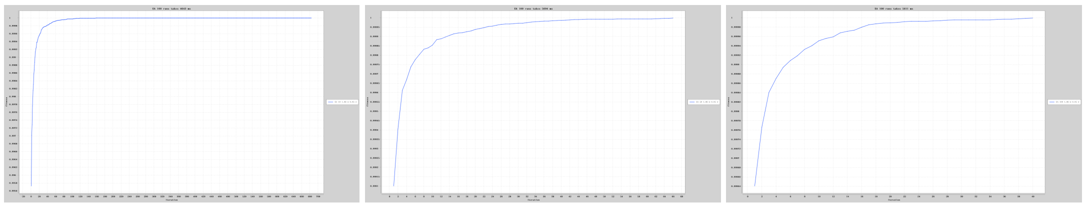


Figure 6.5: Effect of population size on EA experimental results

Population Size	Elites Count	Generation Last	Time Taken(ms)
10	2	682	4643
50	2	66	5694
100	2	40	5622

Table 6.2: Effect of population size on EA experimental results

By increasing the number of population, can increase the number of individuals checked in each generation of Evolutionary algorithm. This can reduce the number of generations needed, but it also increases the time required to check for individuals. So the overall time required is uncertain. As the population size increases, the algorithm can get better solutions faster.

Effect of number of elites on experimental results

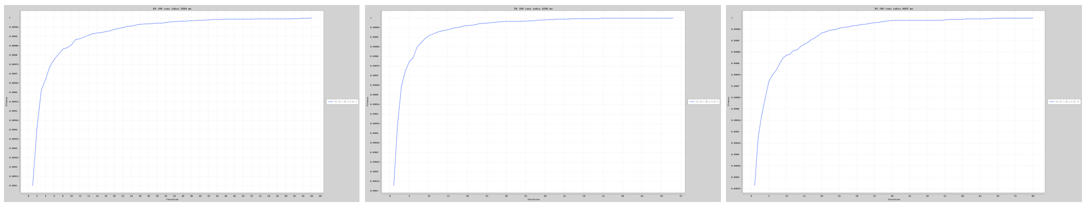


Figure 6.6: Effect of number of elites on EA experimental results

Population Size	Elites Count	Generation Last	Time Taken(ms)
50	2	66	5694
50	5	73	4590
50	10	80	6037

Table 6.3: Effect of number of elites on EA experimental results

The greater the number of elites, the more open the acceptance conditions are. If the number of elites and the population size are the same, it is completely random to find a solution. So the more the number of elites, larger the generation is needed to get the solution, but the running time is controlled by both parameters.

Effect of mutation rate on experimental results

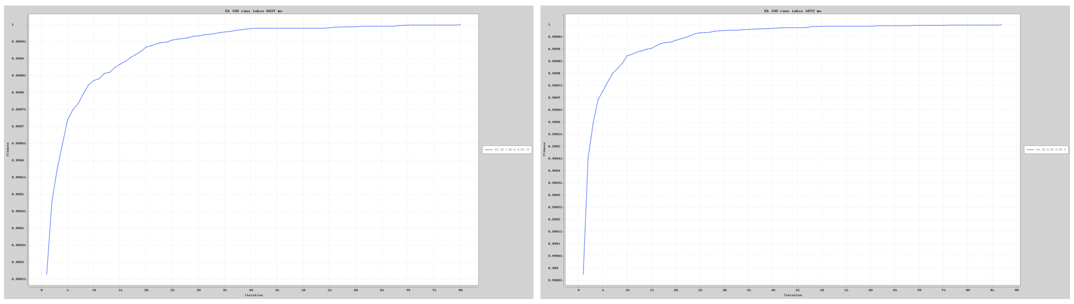


Figure 6.7: Effect of number of elites on EA experimental results

The meaning of mutation and crossover is to ensure the diversity of the population and jump out of the local optimum. In the case of randomly generated jobshop, they will not have a big impact on the experimental results.

6.1.4 Simulated Annealing Algorithm

Effect of initial temperature on experimental results

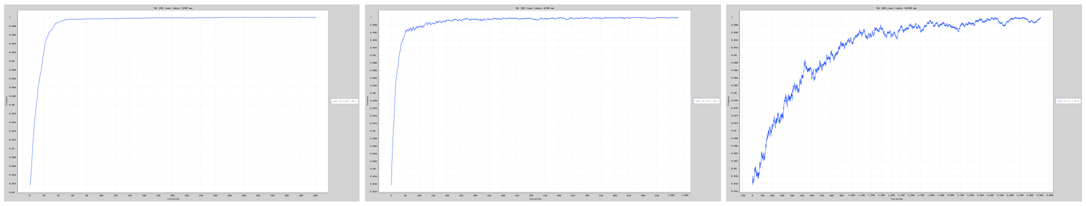


Figure 6.8: Effect of initial temperature on SA experimental results

Initial Temperature	Cooling Rate	Iteration Count	Time Taken(ms)
0.001	0.0001	401	3192
0.01	0.0001	1020	4199
5	0.0001	2909	18100

Table 6.4: Effect of initial temperature on SA experimental results

When the initial temperature is 0.001, the initial acceptance probability is quite small, so the fitness value changes very smoothly, which means that the algorithm can get solution in short time and less iterations[10]. As the initial temperature increases, the probability of accepting a worse solution increases, and it is easier to jump out of the local optimum at the cost of time and iterations.

Effect of cooling rate on experimental results

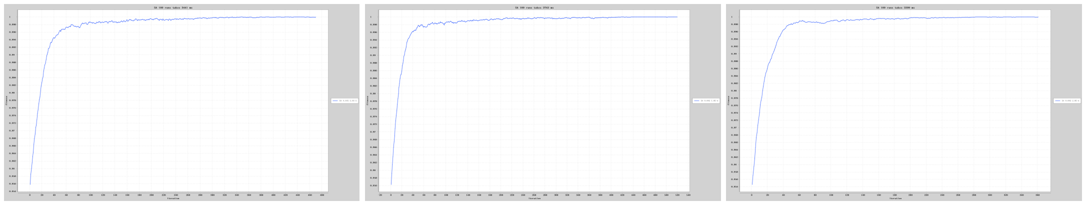


Figure 6.9: Effect of cooling rate on SA experimental results

The system cooling rate controls the rate of decline in acceptance probability. The faster the system cools, the faster the function curve is smoothed.

Initial Temperature	Cooling Rate	Iteration Count	Time Taken(ms)
0.0005	0.00005	470	3441
0.0005	0.0001	519	3743
0.0005	0.0002	362	3399

Table 6.5: Effect of initial temperature on SA experimental results

6.1.5 Strong Selection Weak Mutation regime(SSWM)

Effect of N on experimental results

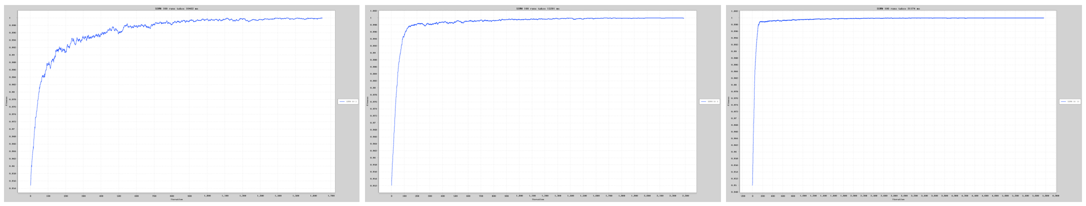


Figure 6.10: Effect of N on experimental SSWM results

N	β	Iteration Count	Time Taken(ms)
2	50	1650	10462
5	50	2290	15291
10	50	5900	31174

Table 6.6: Effect of N on experimental SSWM results

N controls the overall acceptance probability of the worse solution. The larger the N is, the smaller the probability of accepting a more poor solution as a whole, the smoother the fitness value curve. So as N increases, the time and iteration count it takes to get the optimal solution increases.

Effect of β on experimental results

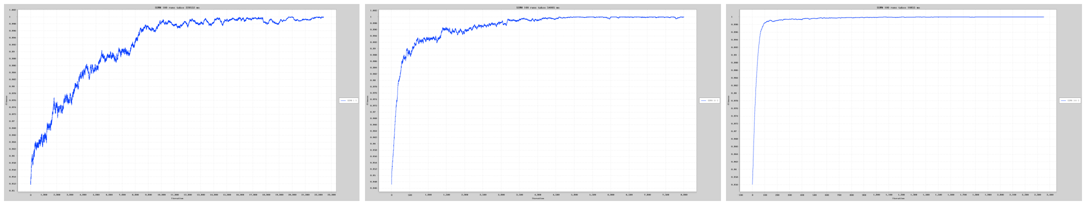


Figure 6.11: Effect of β on SSWM experimental results

N	β	Iteration Count	Time Taken(ms)
5	1	22340	229552
5	10	8000	54005
5	100	2355	10855

Table 6.7: Effect of β on SSWM experimental results

Product of β and $\varDelta f$ should fall in the interval [-1, 3], make the value of acceptance probability function within a suitable range. If β is too small, the probability will be limited to a small change interval around $\frac{1}{N}$. If β is too big, It becomes (1+1)EA.

6.2 All ones jobshop

After analysing the randomly generated jobshop, the following three jobshops use the best performing parameters obtained from previous comparative experiments. And in order to get reasonable experimental results, the population of EA is chosen to be 10.

Result charts

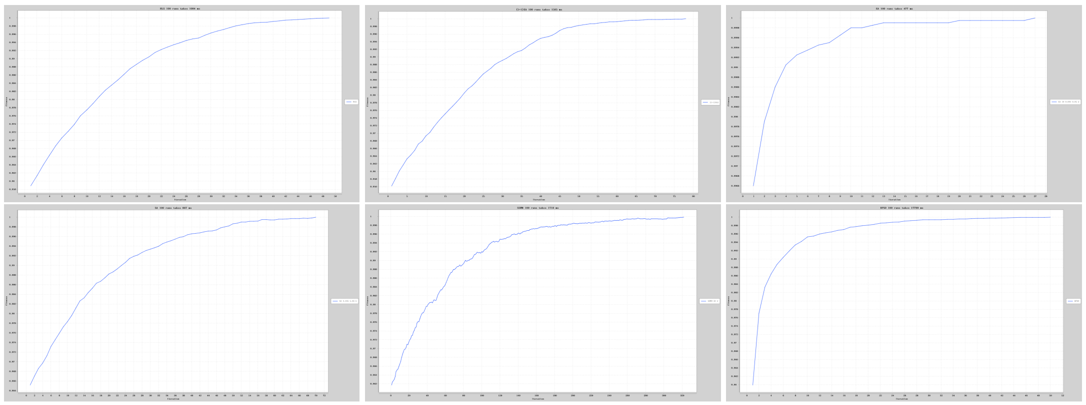


Figure 6.12: All ones jobshop experimental results

Compared to the previously randomly generated jobshop results. The process of finding the optimal solution for all ones jobshop is faster. It is reflected in the number of iterations required by various algorithms and the average calculation time needed is shorter. However, BPSO spend too much time solving the problem, SSWM does not get the optimal solution, and performance of EA is relatively best.

Result table

Algorithm	Time Taken(ms)	Largest Iteration	End fitness
Greedy Algorithm	40	1	1.0
LPT Algorithm	56	1	1.0
RLS Algorithm	1084	49	1.0
(1+1)EA	1505	78	1.0
EA(10, 0.001, 0.95, 2)	477	27	1.0
SA(0.005, 0.00005)	867	70	1.0
SSWM(50, 2)	1718	323	0.99985
BPSO	17799	50	1.0

Table 6.8: All ones jobshop experimental results

6.3 Linearly increased jobshop

Determine the algorithm parameters based on the analysis of the randomly generated jobshop.

Result charts

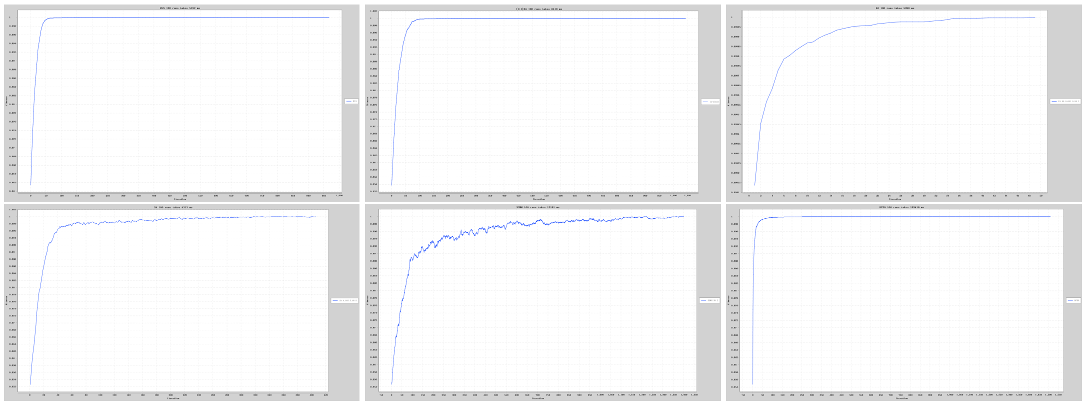


Figure 6.13: Linearly increased jobshop experimental results

On this issue, most algorithms can quickly find a solution that is closer to the optimal solution, but it takes a lot of time to get the approximate optimal solution. Although SA did not find a solution that is closer to the optimal solution quickly, but SA used the shortest time to find the approximate optimal solution. In general, the algorithm performance of EA and SA is better than the others.

Result table

Algorithm	Time Taken(ms)	Largest Iteration	End fitness
Greedy Algorithm	42	1	0.9975062344139651
LPT Algorithm	53	1	1.0
RLS Algorithm	5192	967	0.9999997506234414
(1+1)EA	6410	1044	0.9999995012468826
EA(50, 0.001, 0.95, 2)	5098	49	0.9999992518703241
SA(0.005, 0.00005)	4313	406	0.9999992518703241
SSWM(50, 2)	13181	1401	0.9999855361596011
BPSO	195416	1511	0.9999997506234414

Table 6.9: Linearly increased jobshop experimental results

6.4 Worst case

Determine the algorithm parameters based on the analysis of the randomly generated jobshop, and in order to get reasonable experimental results, the population of EA is chosen to be 20.

Result charts

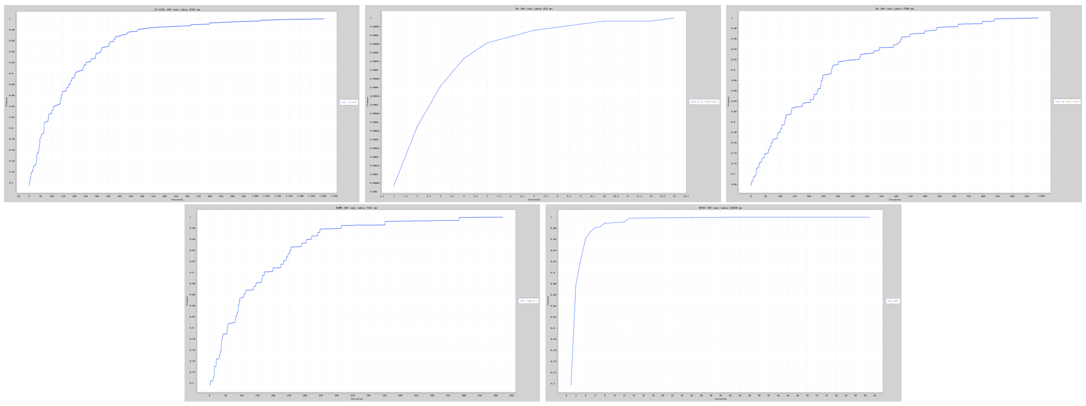


Figure 6.14: Worst case jobshop experimental results

When testing the worst case jobshop, the fitness values of various algorithms generally rise in a stepwise manner. Although both SA and SSWM accept a more poor solution, they are not obvious in the result image. The deficiencies of greedy algorithm and RLS are also reflected. In the worst case, the result of greedy algorithm is unacceptable, and RLS also falls into local optimum. However EA's performance is outstanding, both running time and iteration count are much less than other algorithms.

Result table

Algorithm	Time Taken(ms)	Largest Iteration	End fitness
Greedy Algorithm	40	1	0.6779964221824687
LPT Algorithm	51	1	1.0
RLS Algorithm	Fall	into	local optimum
(1+1)EA	6581	1307	1.0
EA(20, 0.001, 0.95, 2)	653	13	1.0
SA(0.005, 0.00005)	7290	990	1.0
SSWM(50, 2)	7421	923	0.9999463327370305
BPSO	22420	63	1.0

Table 6.10: Worst case jobshop experimental results

Chapter 7

Conclusions

The purpose of this dissertation is to solve the makespan scheduling problem through greedy algorithm, LPT algorithm, and various randomized search heuristic algorithms, including RLS, (1+1)EA, EA, BPSO, SA, and SSWM. Record and analyse the performance of various algorithms on solving this problem.

Among the EA, SA, and SSWM algorithms, different combinations of parameters affect the performance of the algorithm to solve the problem. Before comparing with other algorithms, it is necessary to test different combinations of parameters to get the best possible parameters combination and represent the algorithm for comparison.

In order to obtain reliable data and compare the functions of different algorithms, I strictly perform requirements analysis, system design, implementation and testing. Finally, Java programming is implemented, and testers can customize jobshop, algorithm parameters, and number of algorithm runs, in order to get different experimental results.

In the process of solving the makepan problem, for allones, linear, and random jobshops. The performance of EA and SA(0.005, 0.00005) is better than other algorithms. EA can find the relative optimal solution with the least number of iterations. SA can find the relative optimal solution in the shortest time. BPSO take too much time for single iteration, and the overall performance is not as good as other algorithms. For the worst case, the performance of EA is significantly better than other algorithms, reflected in the running time and iteration takes. RLS always falls into local optimum in 100 runs test, and the greedy algorithm solution is unacceptable.

7.1 Limitation and Further work

For this dissertation, there are some unresolved issues, project improvements, and content that deserves further study. Details are as follows:

- The understanding of the lemma and theorem in the "Existing work on EA for PARTITION" is not sufficient. And needs to be further understood through experimental verification.

- For algorithms with parameters, before schedule different jobshops, use more parameter combinations, observe the influence of parameter changes on the algorithm results. Record data, and draw the curve to find the relative optimal parameter combination.

- This dissertation is intended for testers who are familiar with the code. There is no user-friendly interfaces, so I can design a mobile app or a website based on this. Users who are unfamiliar with the random search heuristic algorithm can easily understand these algorithms by changing parameters themselves and feeling the result changes.
- This dissertation only use random search heuristic algorithms to solve makespan scheduling, and there are many interesting problems that can be solved by random search heuristic. For example, EA uses polygons to design a car, or generate famous painting "Mona Lisa".

Bibliography

[1] Auger, A., and Witt, C. Theory of randomized search heuristics. *Algorithmica 64*, 4 (Dec 2012), 621–622.

[2] Chopra, S., and Rao, M. R. The partition problem. *Mathematical Programming 59*, 1-3 (1993), 87–115.

[3] Eiben, A. E., Smith, J. E., et al. *Introduction to evolutionary computing*, vol. 53. Springer, 2003.

[4] Feng, W., Zheng, L., and Li, J. Scheduling policies in multi-product manufacturing systems with sequence-dependent setup times. In *Proceedings of the Winter Simulation Conference* (2011), WSC '11, Winter Simulation Conference, pp. 2055–2066.

[5] Fortnow, L. The status of the p versus np problem. *Commun. ACM 52*, 9 (Sept. 2009), 78–86.

[6] Jiang, L., and Eberlein, A. Towards a framework for understanding the relationships between classical software engineering and agile methodologies. In *Proceedings of the 2008 International Workshop on Scrutinizing Agile Practices or Shoot-out at the Agile Corral* (New York, NY, USA, 2008), APOS '08, ACM, pp. 9–14.

[7] Kamal, M., Afzali-Kusha, A., Safari, S., and Pedram, M. Ople: A heuristic custom instruction selection algorithm based on partitioning and local exploration of application dataflow graphs. *ACM Trans. Embed. Comput. Syst. 14*, 4 (Sept. 2015), 72:1–72:23.

[8] Kennedy, J., and Eberhart, R. C. A discrete binary version of the particle swarm algorithm. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation* (Oct 1997), vol. 5, pp. 4104–4108 vol.5.

[9] Laporte, G. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research 59*, 3 (1992), 345–358.

[10] Liu, Y., Zhang, D., and Leung, S. C. A simulated annealing algorithm with a new neighborhood structure for the timetabling problem. In *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation* (New York, NY, USA, 2009), GEC '09, ACM, pp. 381–386.

[11] M, J., and Iyer, S. A method to construct counterexamples for greedy algorithms. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2012), ITiCSE '12, ACM, pp. 238–243.

[12] Neumann, F., and Witt, C. Bioinspired computation in combinatorial optimization: Algorithms and their computational complexity. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation* (2013), ACM, pp. 567–590.

[13] Orlin, J. B. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming 118*, 2 (2009), 237–251.

[14] Paixão, T., Heredia, J. P., Sudholt, D., and Trubenová, B. Towards a runtime comparison of natural and artificial evolution. *Algorithmica 78*, 2 (2017), 681–713.

[15] Staff, S. E. N. Abstracts in software engineering. *SIGSOFT Softw. Eng. Notes 8*, 1 (Jan. 1983), 99–102.

[16] Sudholt, D., and Witt, C. Runtime analysis of a binary particle swarm optimizer. *Theor. Comput. Sci. 411*, 21 (May 2010), 2084–2100.

[17] techopedia. What is risk analysis. `https://www.techopedia.com/definition/16522/risk-analysis`.

[18] Trafalgar. Ethical, environmental and legal issues. `https://sites.google.com/a/trafalgarconnected.com/computer-science/computer-science/ocr-gcse-computing/computing-fundamentals/ethical-environmental-and-legal-issues`.

[19] Van Laarhoven, P. J., and Aarts, E. H. Simulated annealing. In *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.

[20] Weisstein, E. W. "np-problem." from mathworld–a wolfram web resource. `http://mathworld.wolfram.com/NP-Problem.html`.

Appendices

Appendix A

Experimental result images

A.1 Randomly Generated Jobshop

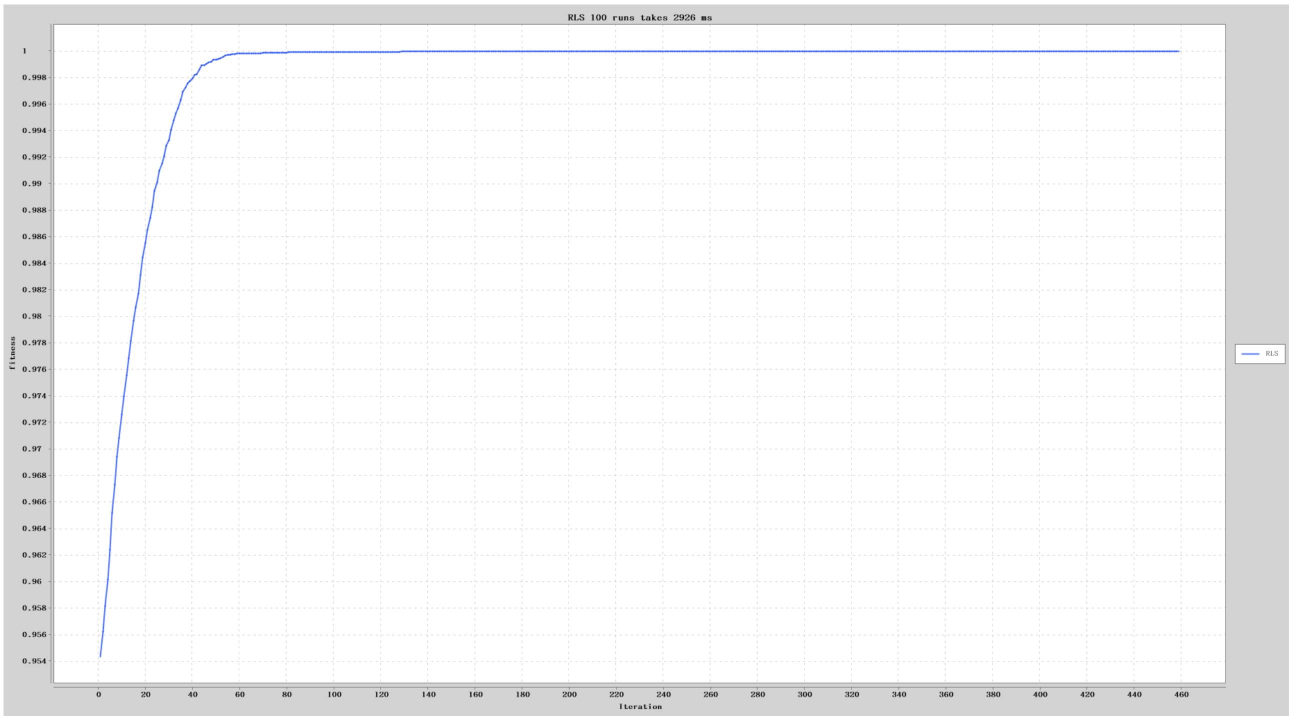


Figure A.1: RLS randomly generated

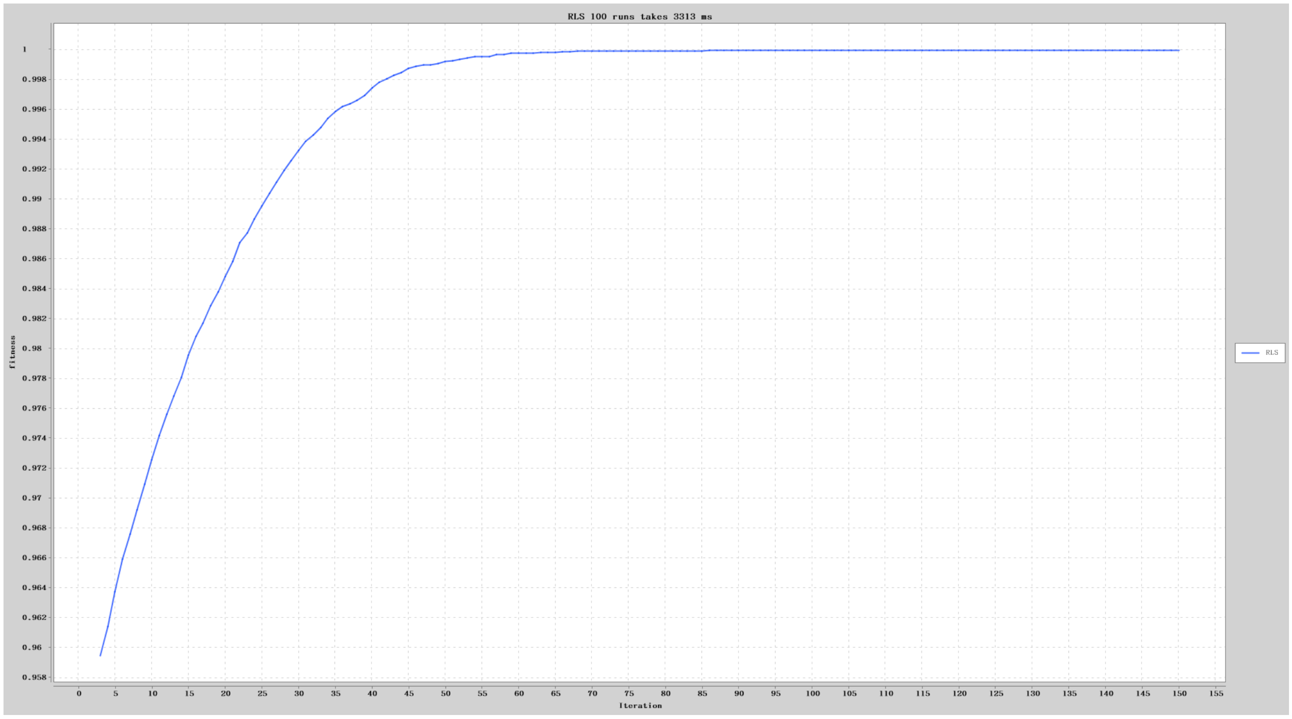


Figure A.2: RLS randomly generate, cut [2, 150] iterations

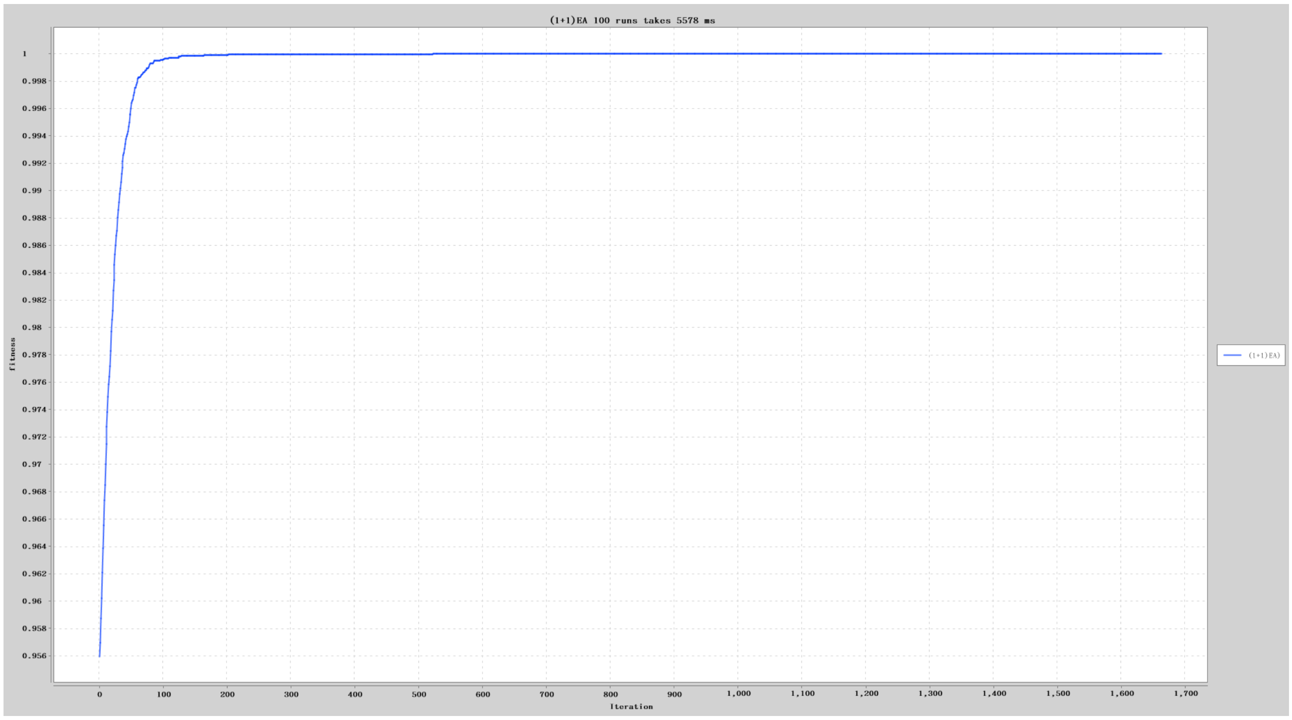


Figure A.3: (1+1)EA randomly generated

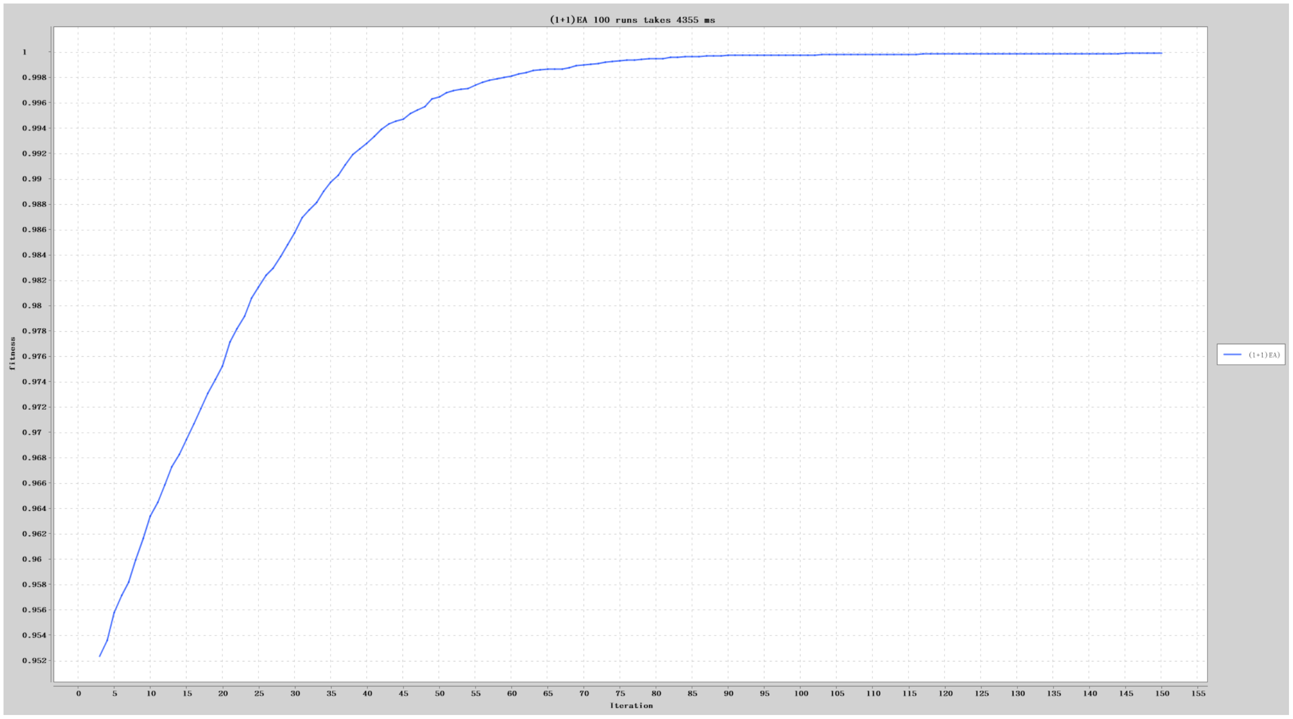


Figure A.4: (1+1)EA randomly generate, cut [2, 150] iterations

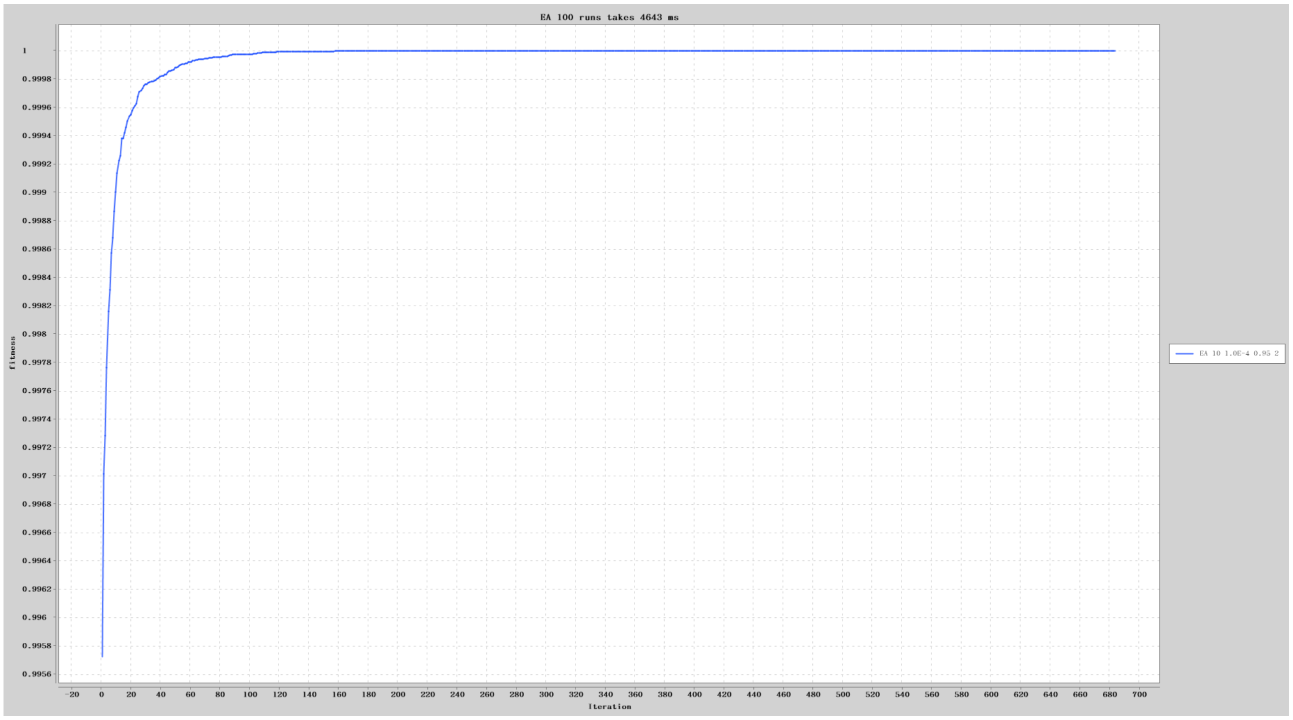


Figure A.5: EA randomly generate, *populationSize* : 10, *elitismCount* : 2

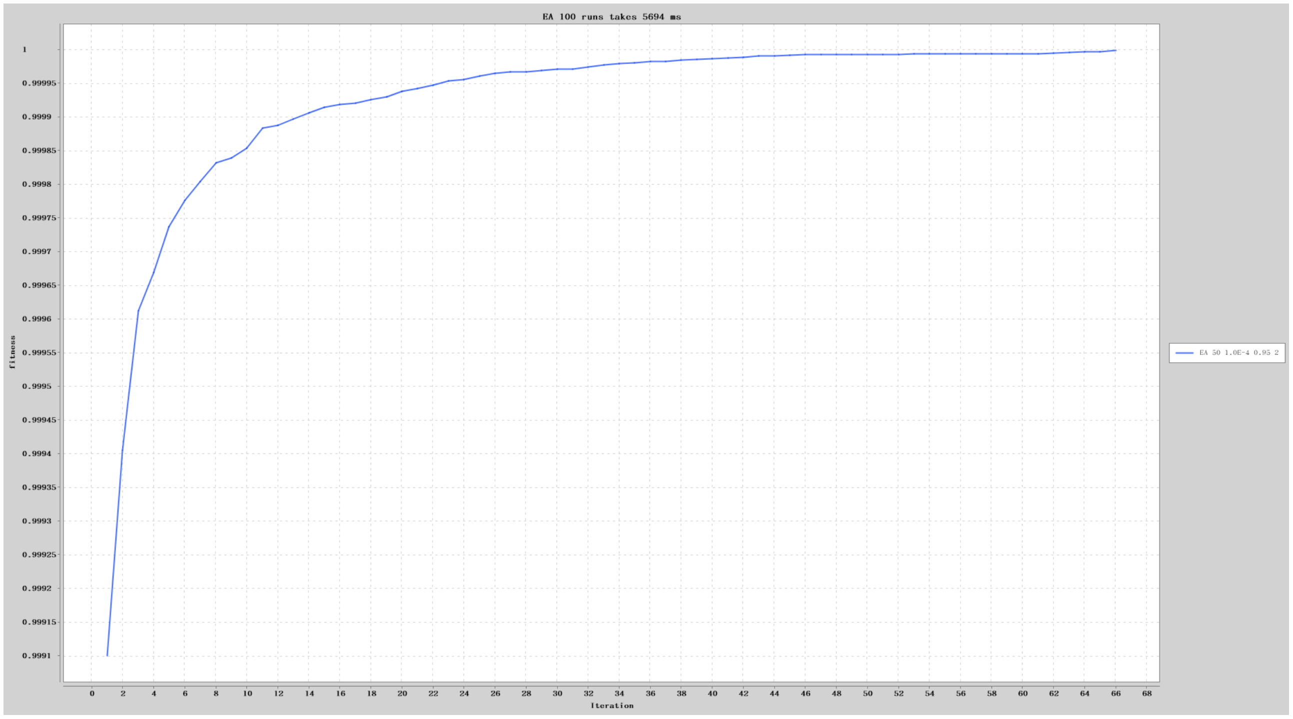


Figure A.6: EA randomly generate, $populationSize$: 50, $elitismCount$: 2

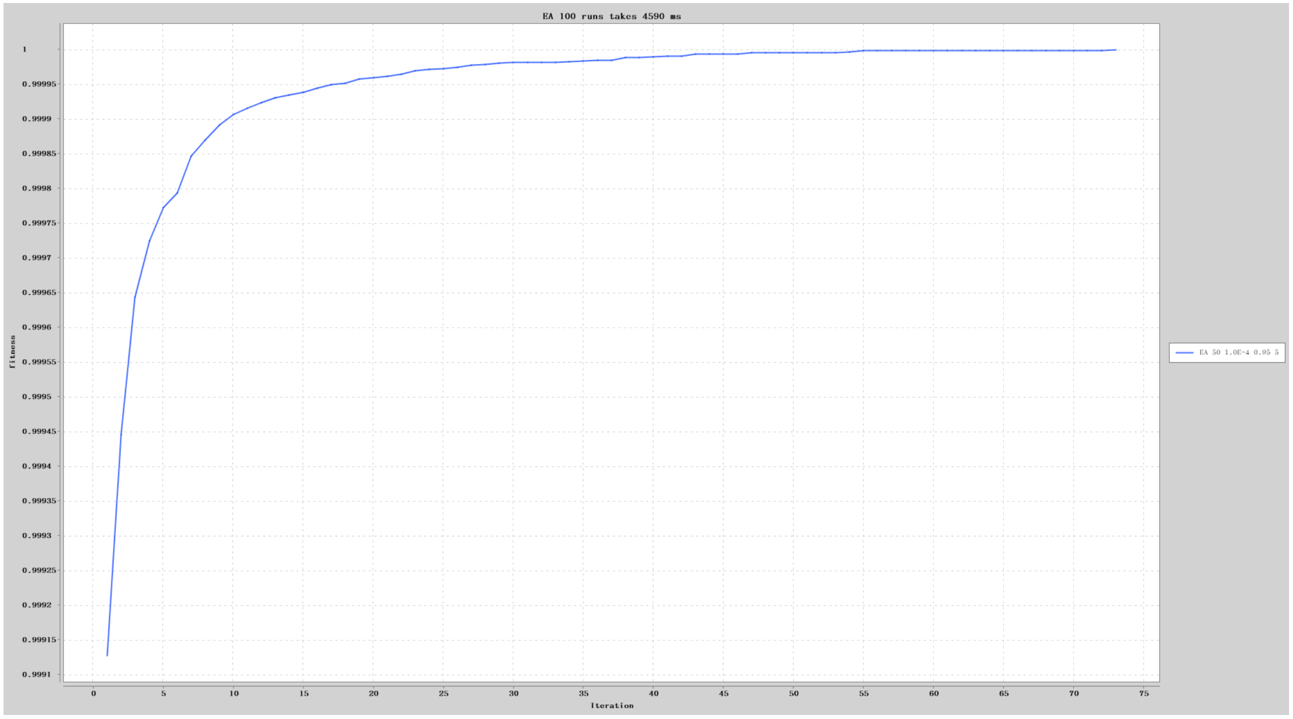


Figure A.7: EA randomly generate, $populationSize$: 50, $elitismCount$: 5

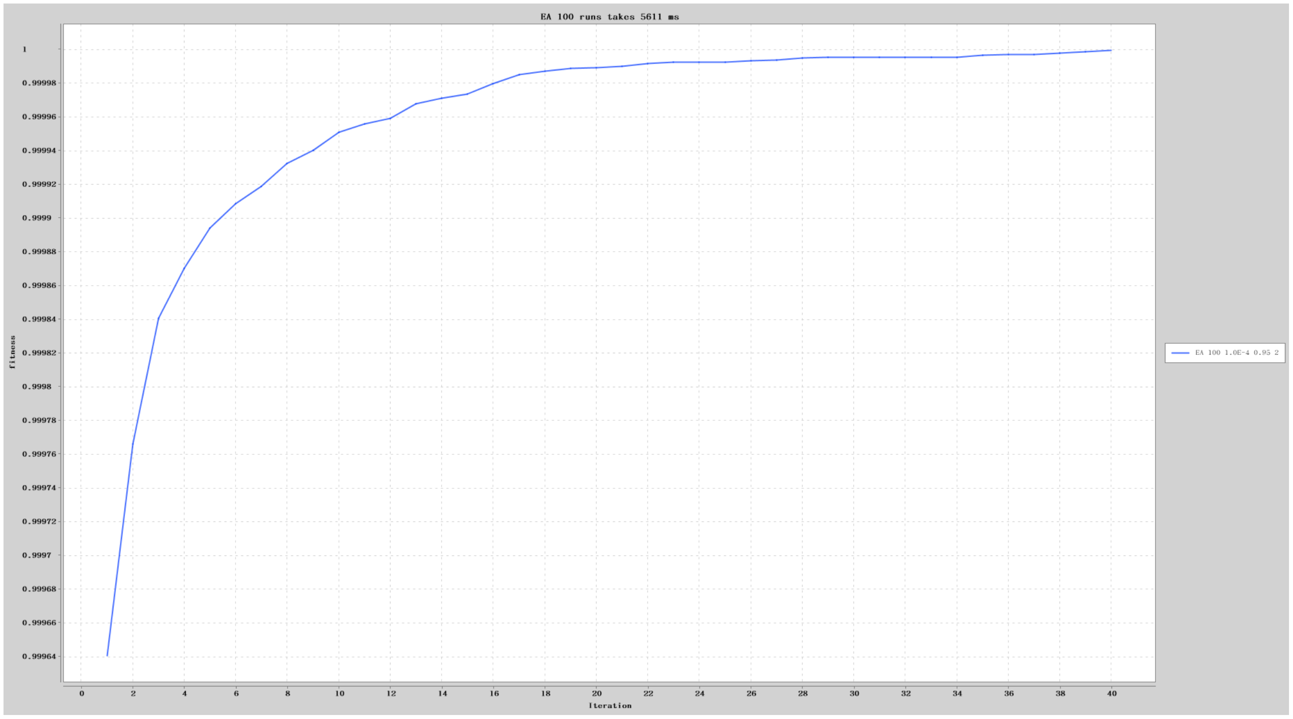


Figure A.8: EA randomly generate, *populationSize* : 100, *elitismCount* : 2

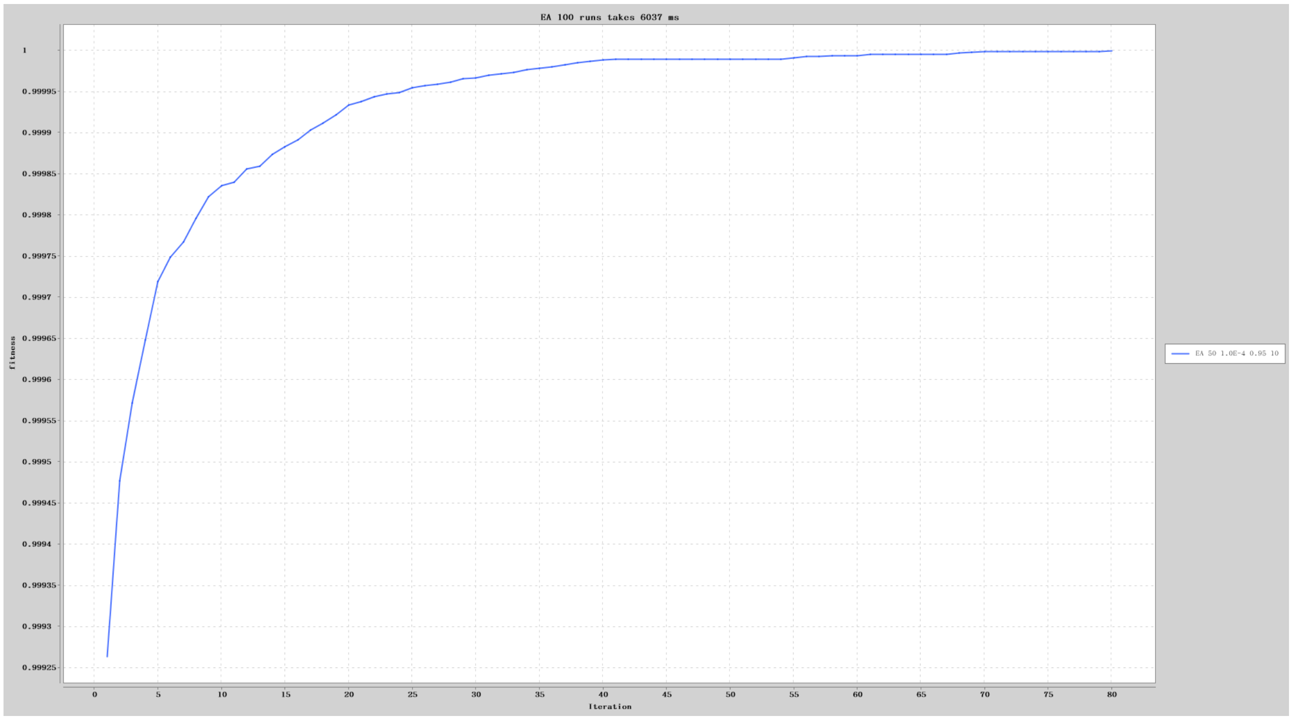


Figure A.9: EA randomly generate, *populationSize* : 50, *elitismCount* : 10

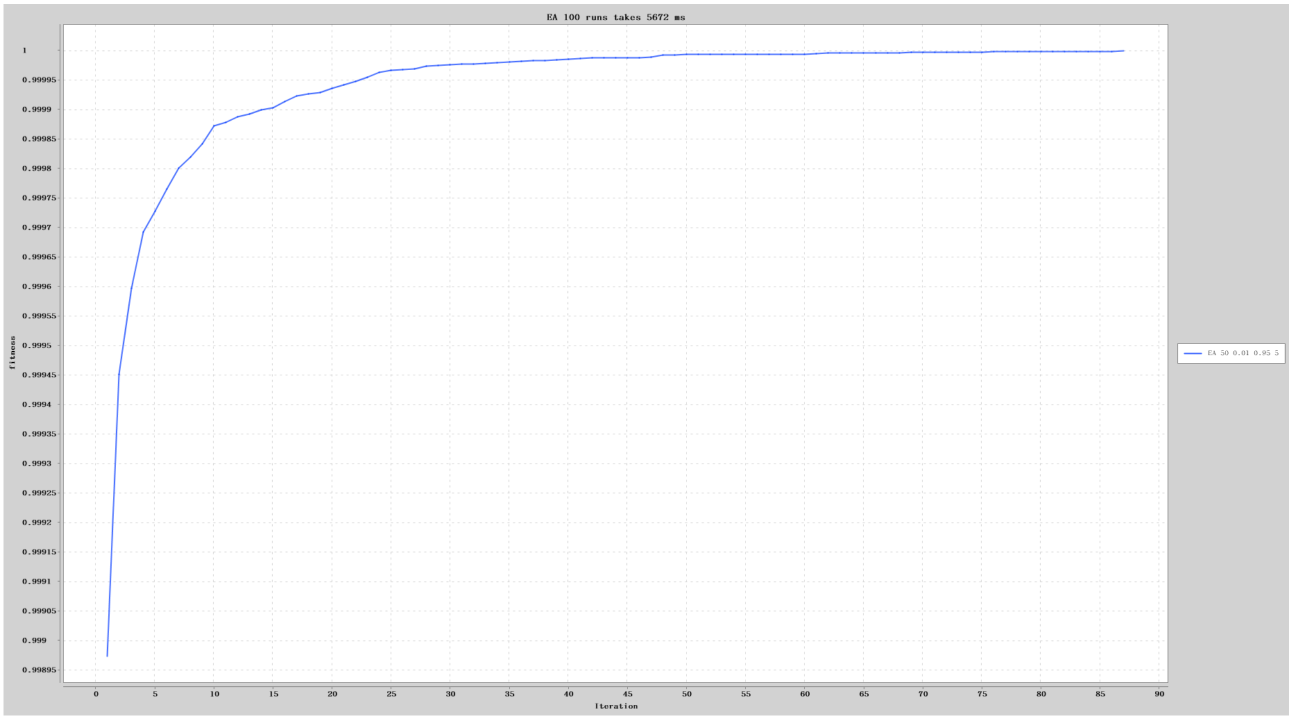


Figure A.10: EA randomly generate, $populationSize$: 50, $elitismCount$: 10, $mutationRate$: 0.01

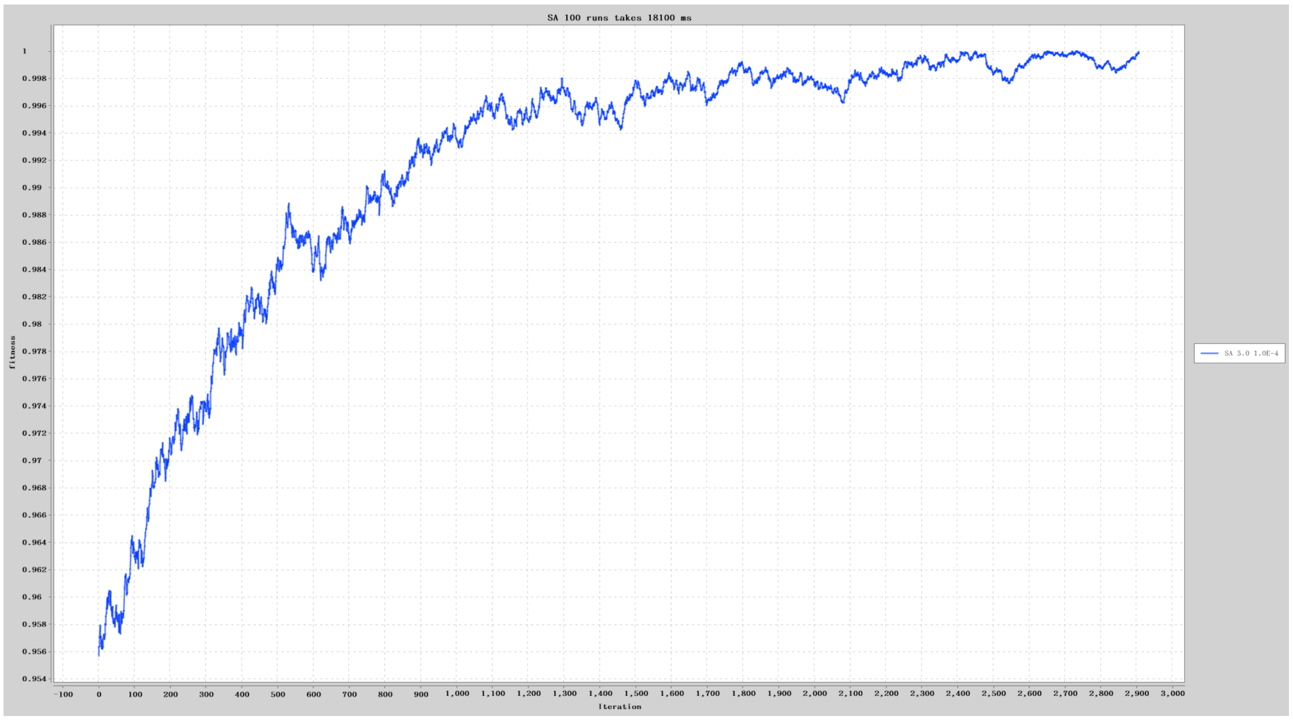


Figure A.11: SA randomly generate, $init\_temp$: 5, $coolingRate$: 0.0001

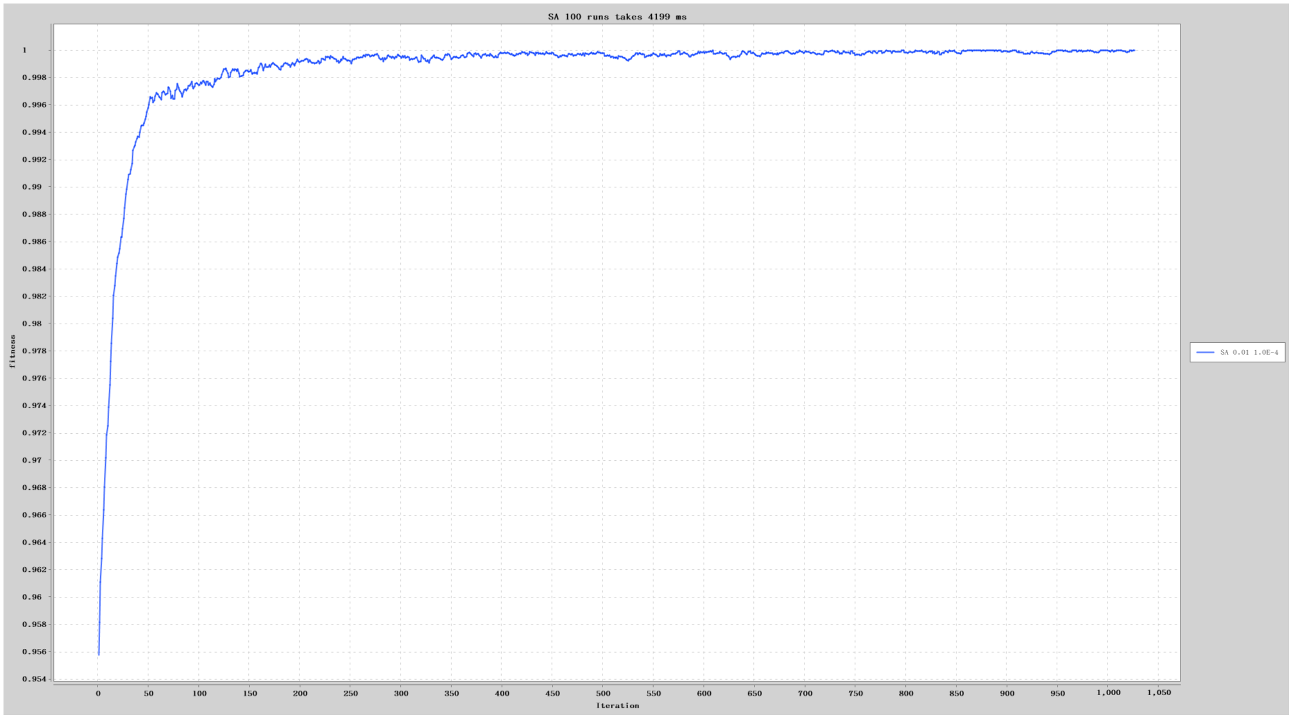


Figure A.12: SA randomly generate, $init\_temp$: 0.01, $coolingRate$: 0.0001

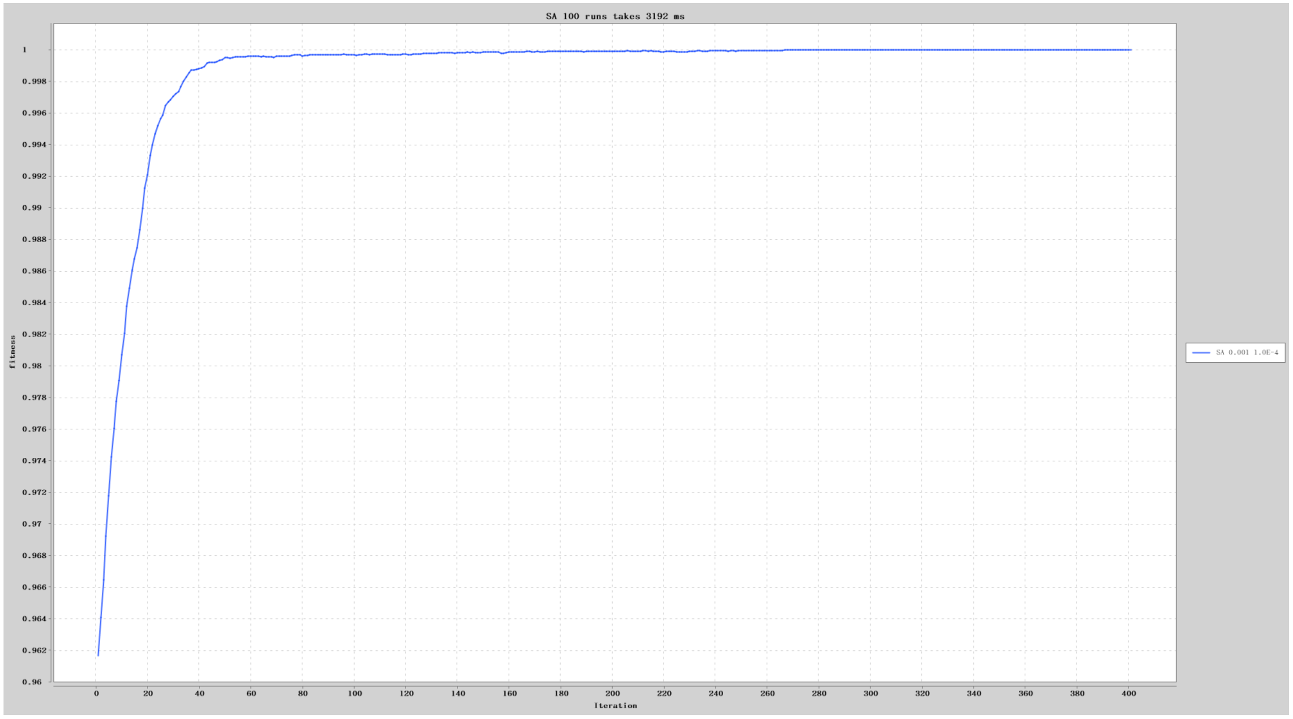


Figure A.13: SA randomly generate, $init\_temp$: 0.001, $coolingRate$: 0.0001

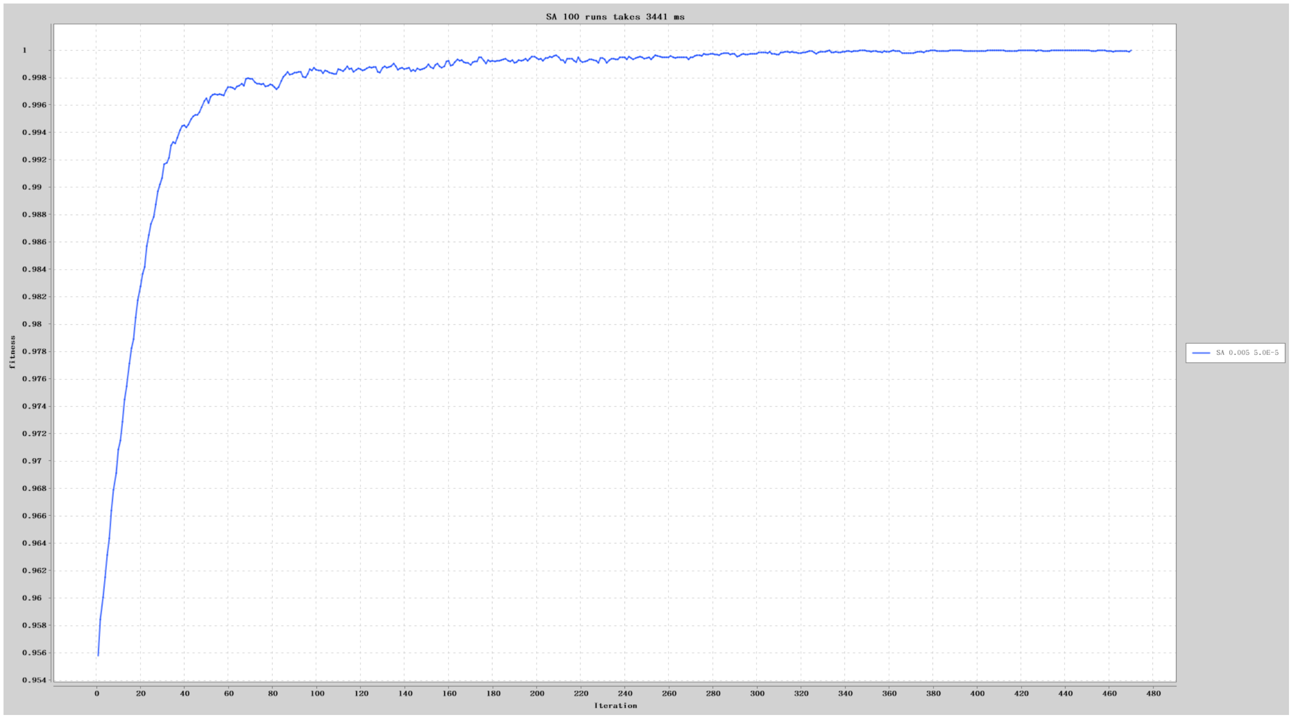


Figure A.14: SA randomly generate, *init_temp* : 0.005, *coolingRate* : 0.00005

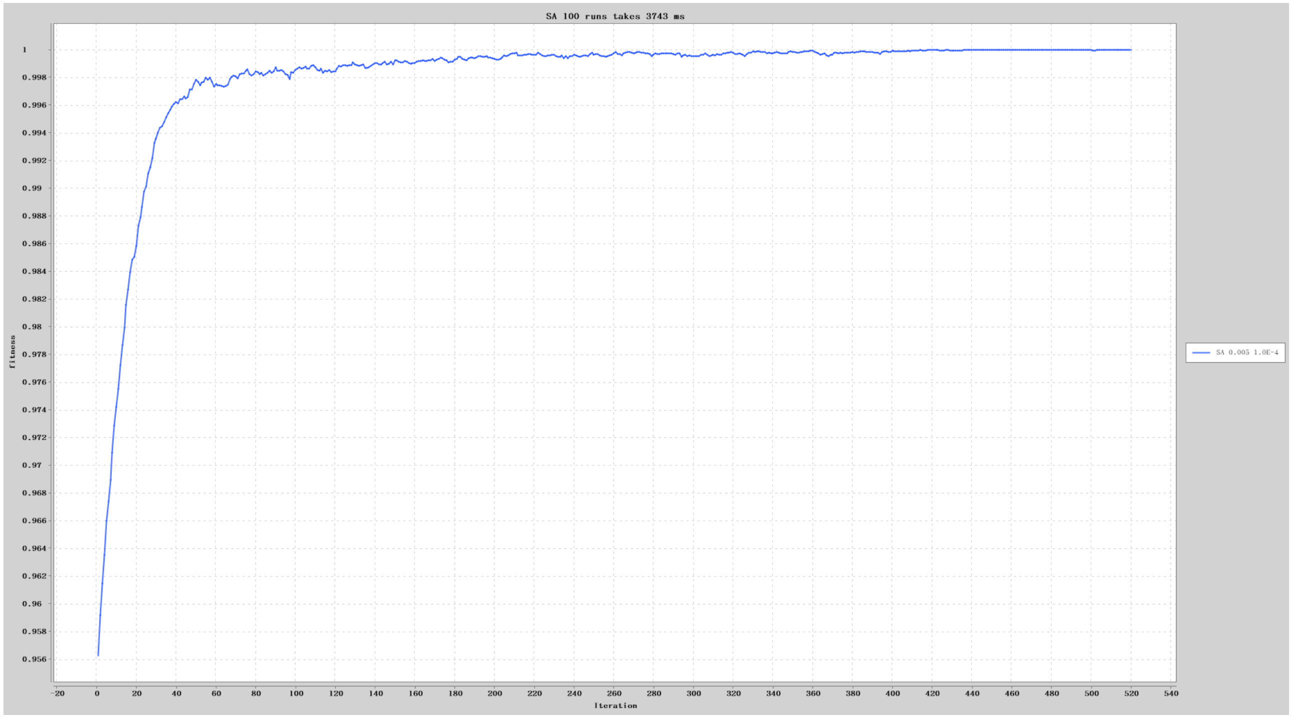


Figure A.15: SA randomly generate, *init_temp* : 0.005, *coolingRate* : 0.0001

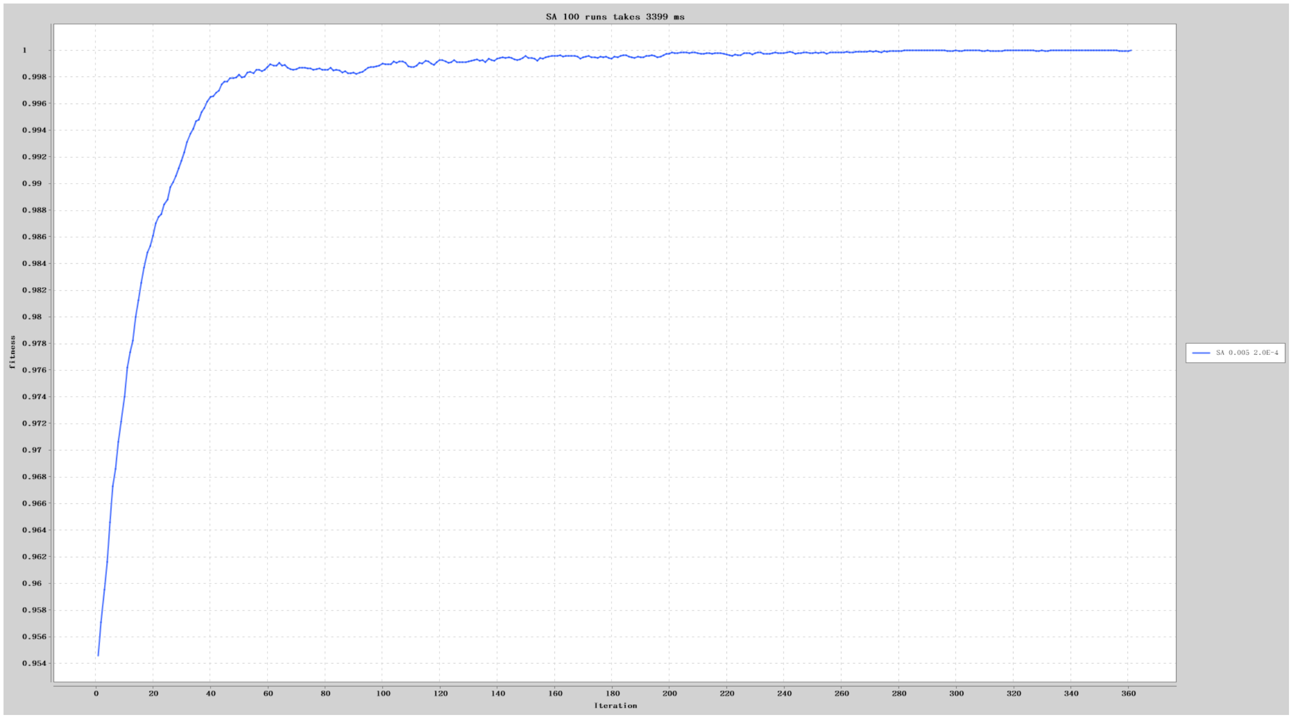


Figure A.16: SA randomly generate, $init\_temp$: 0.005, $coolingRate$: 0.0002

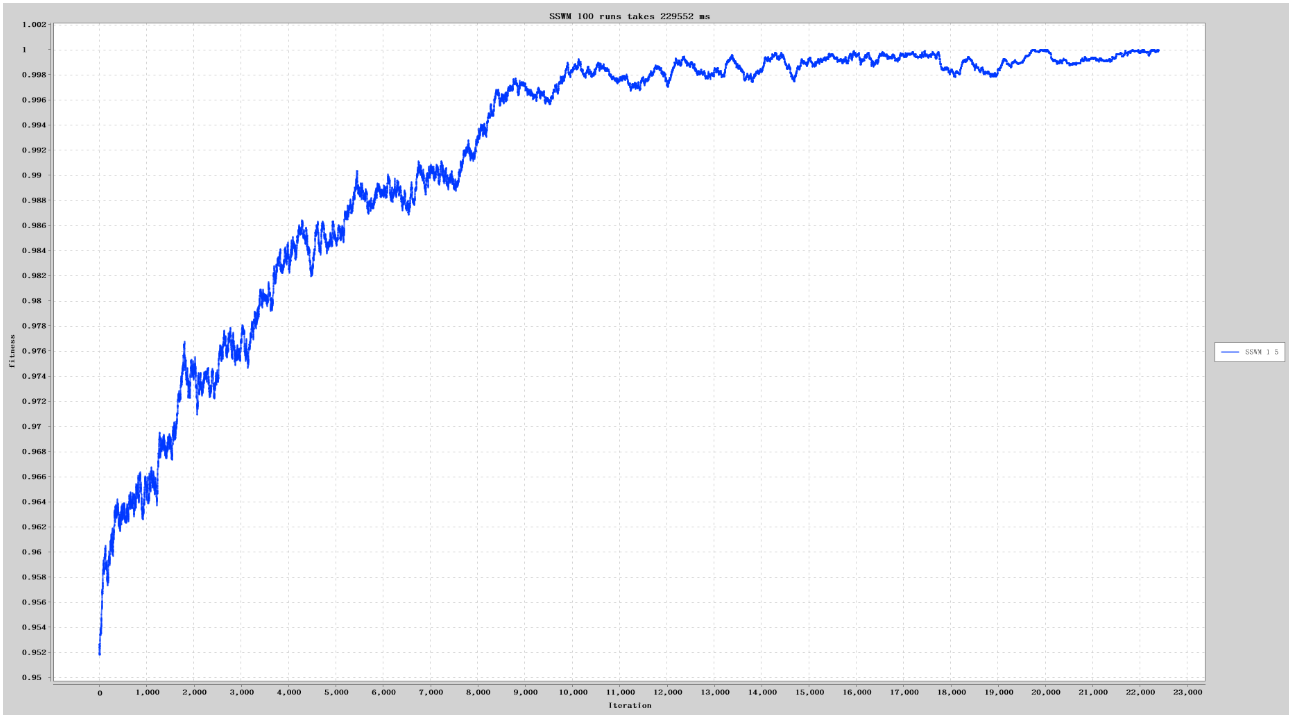


Figure A.17: SSWM randomly generate, N : 5, β : 1

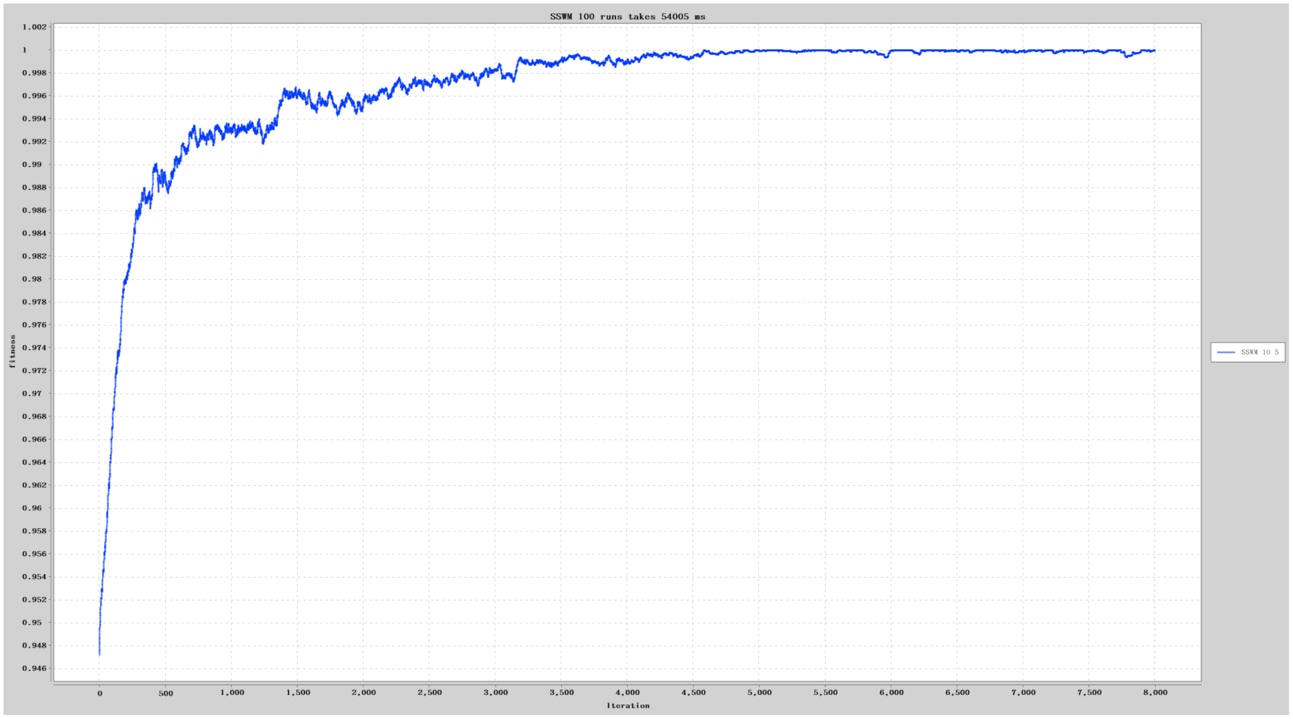


Figure A.18: SSWM randomly generate, N : 5, β : 10

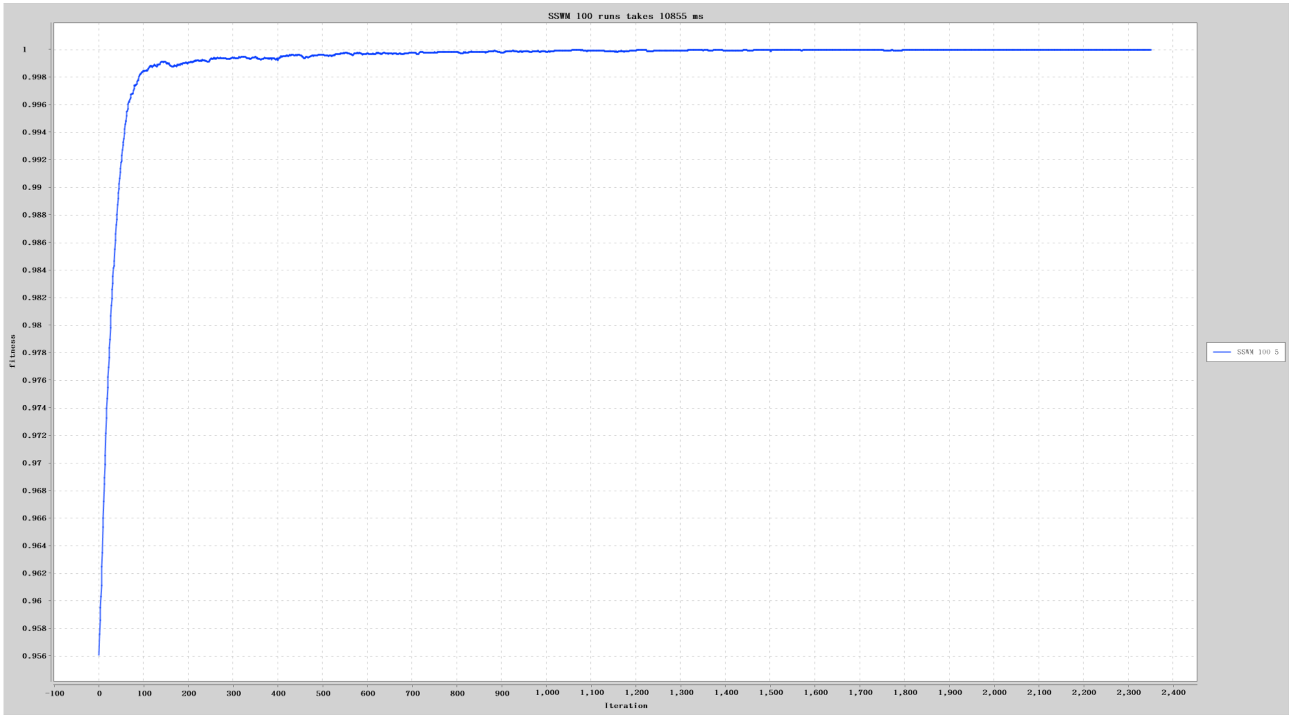


Figure A.19: SSWM randomly generate, N : 5, β : 100

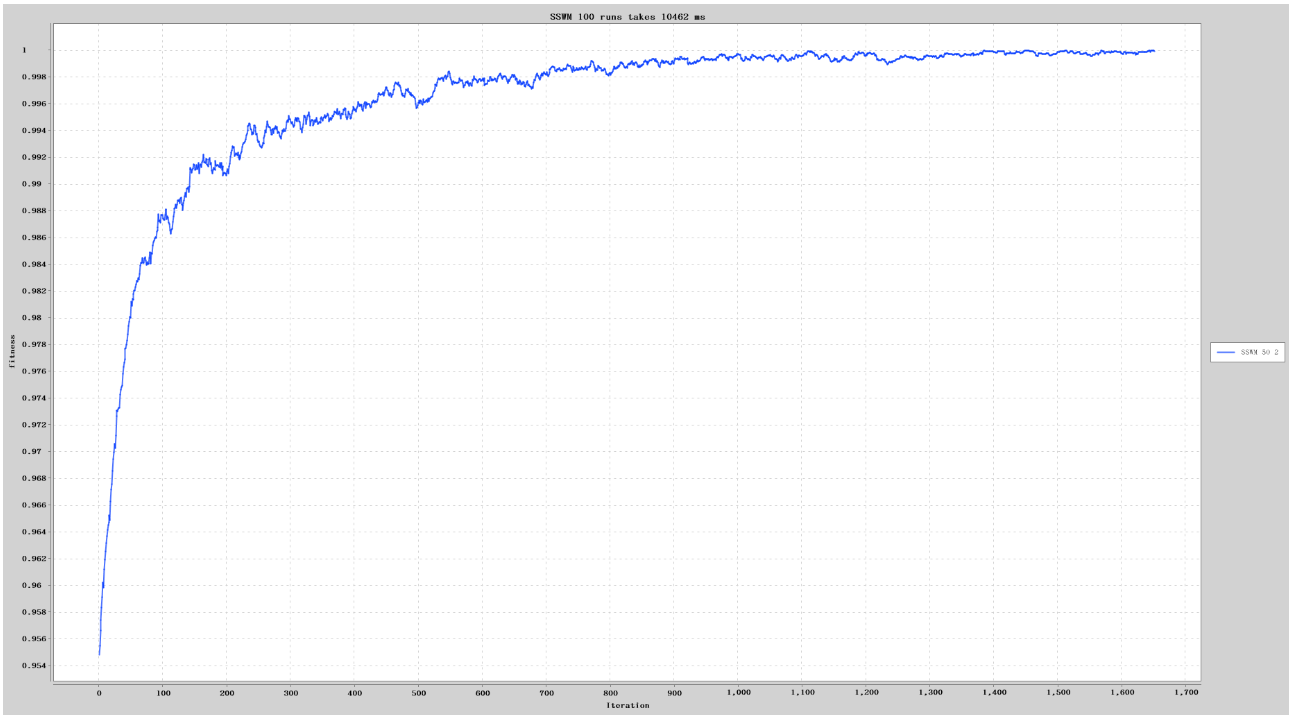


Figure A.20: SSWM randomly generate, N : 2, β : 50

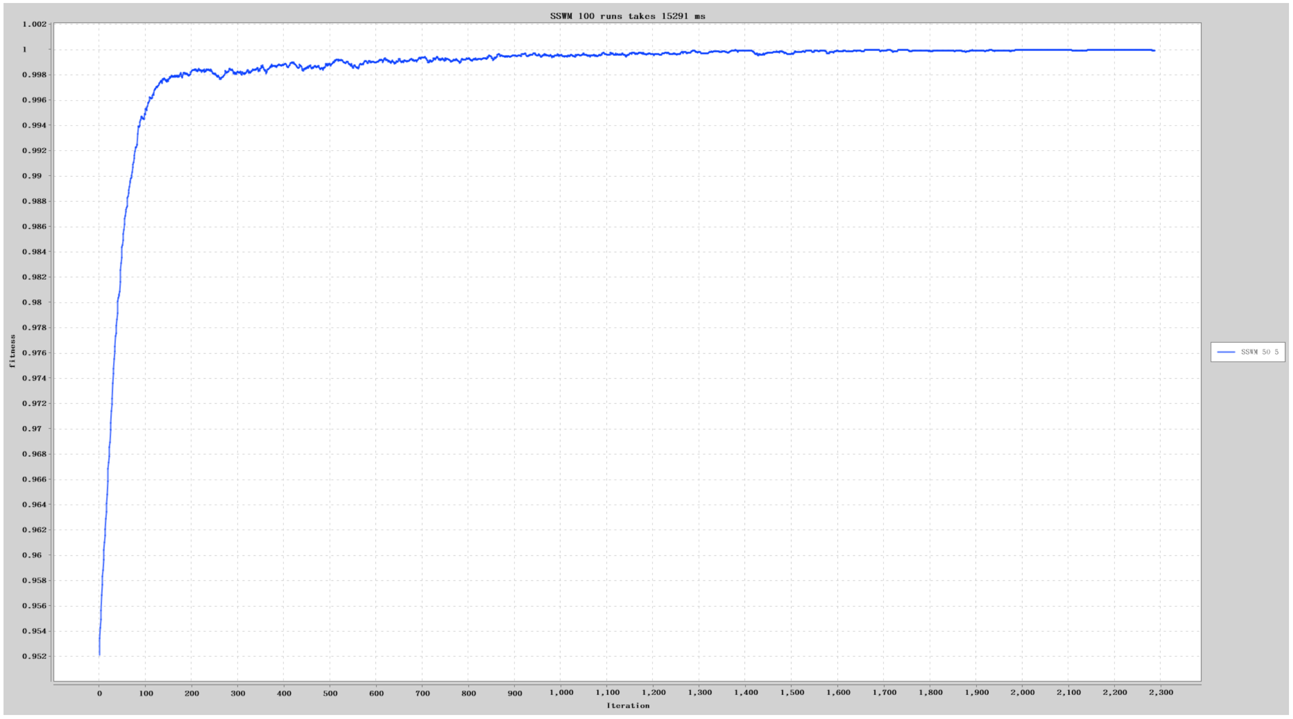


Figure A.21: SSWM randomly generate, N : 5, β : 50

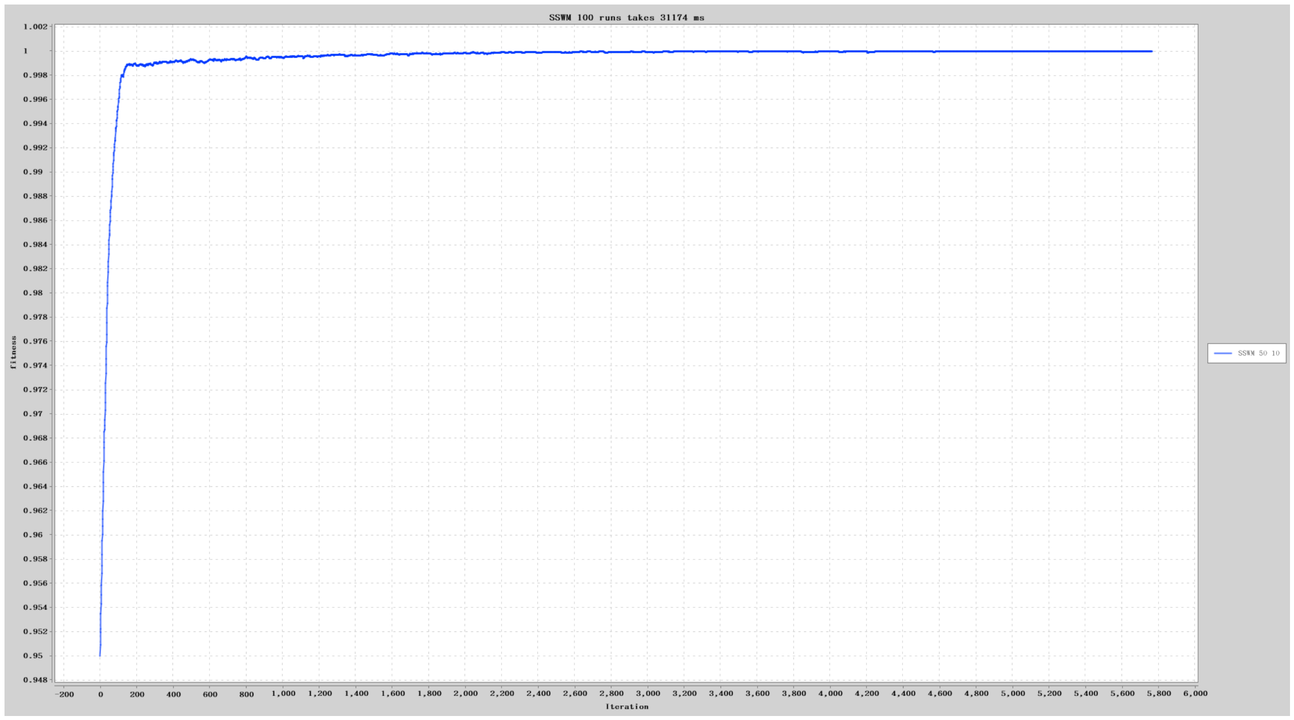


Figure A.22: SSWM randomly generate, N : 10, β : 50

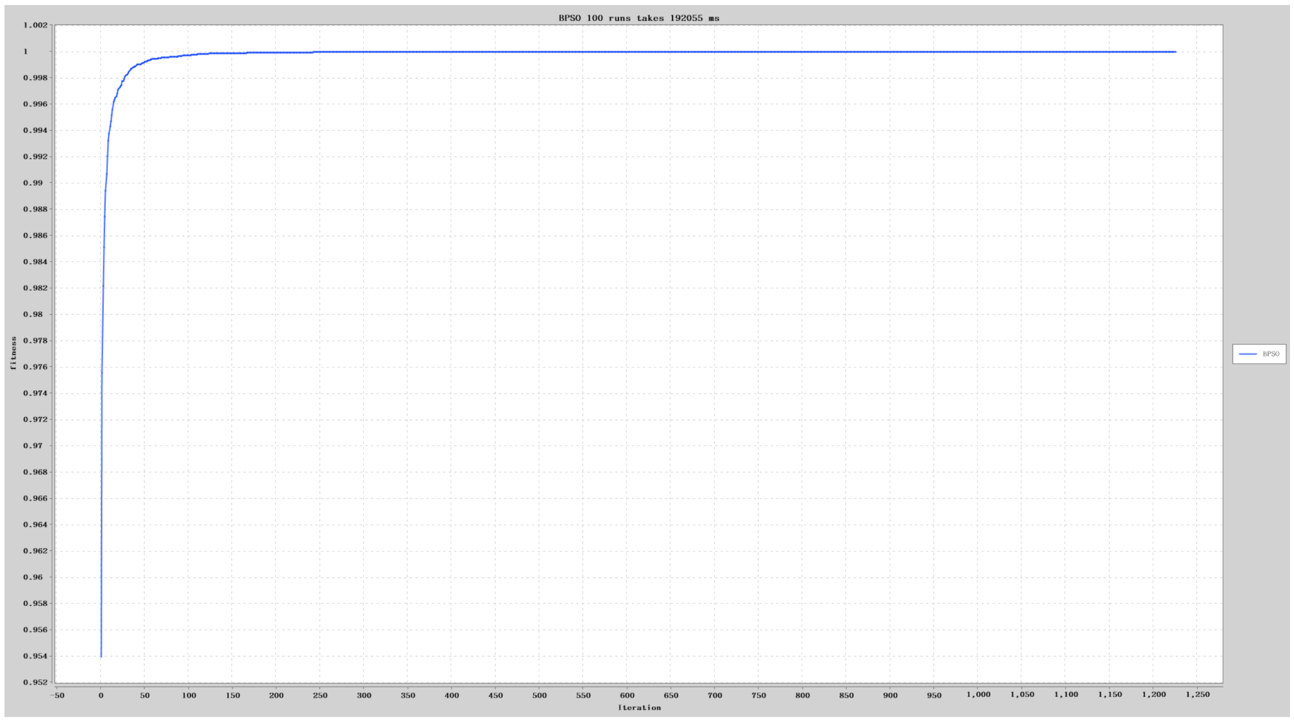


Figure A.23: BPSO randomly generated

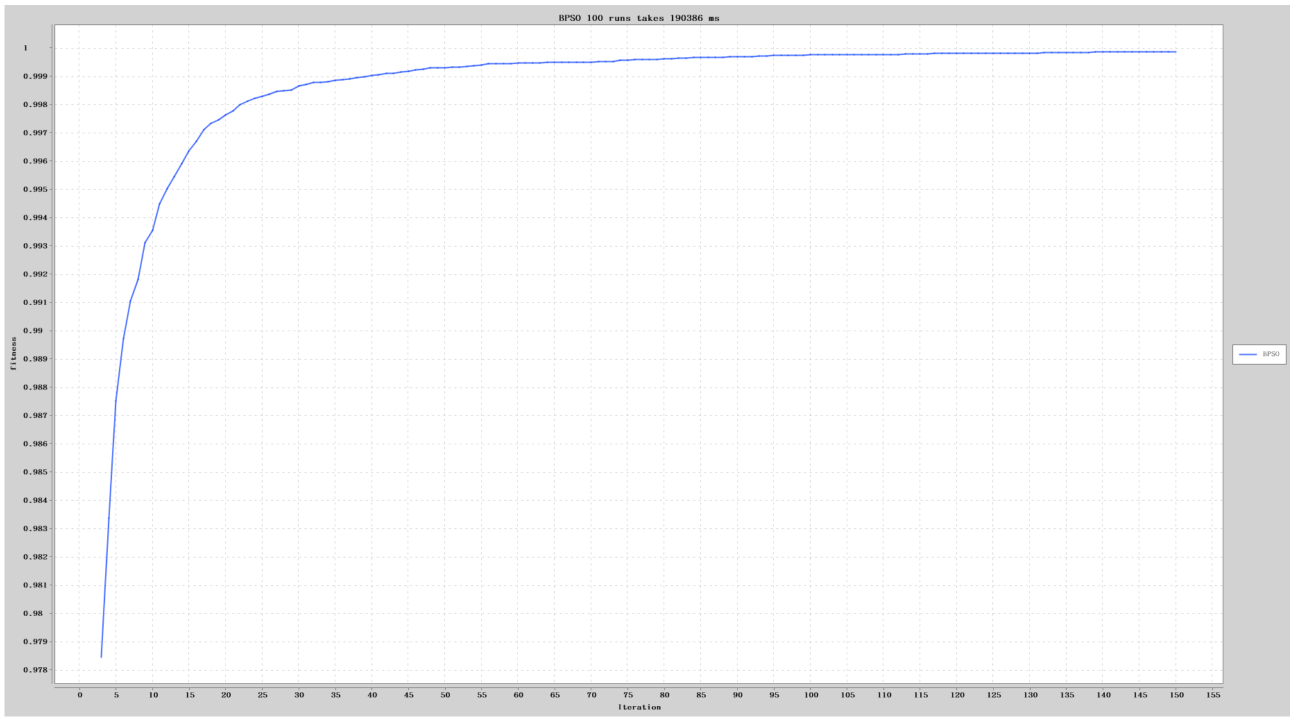


Figure A.24: BPSO randomly generate, cut [2, 150] iterations

A.2 All Ones Jobshop Result

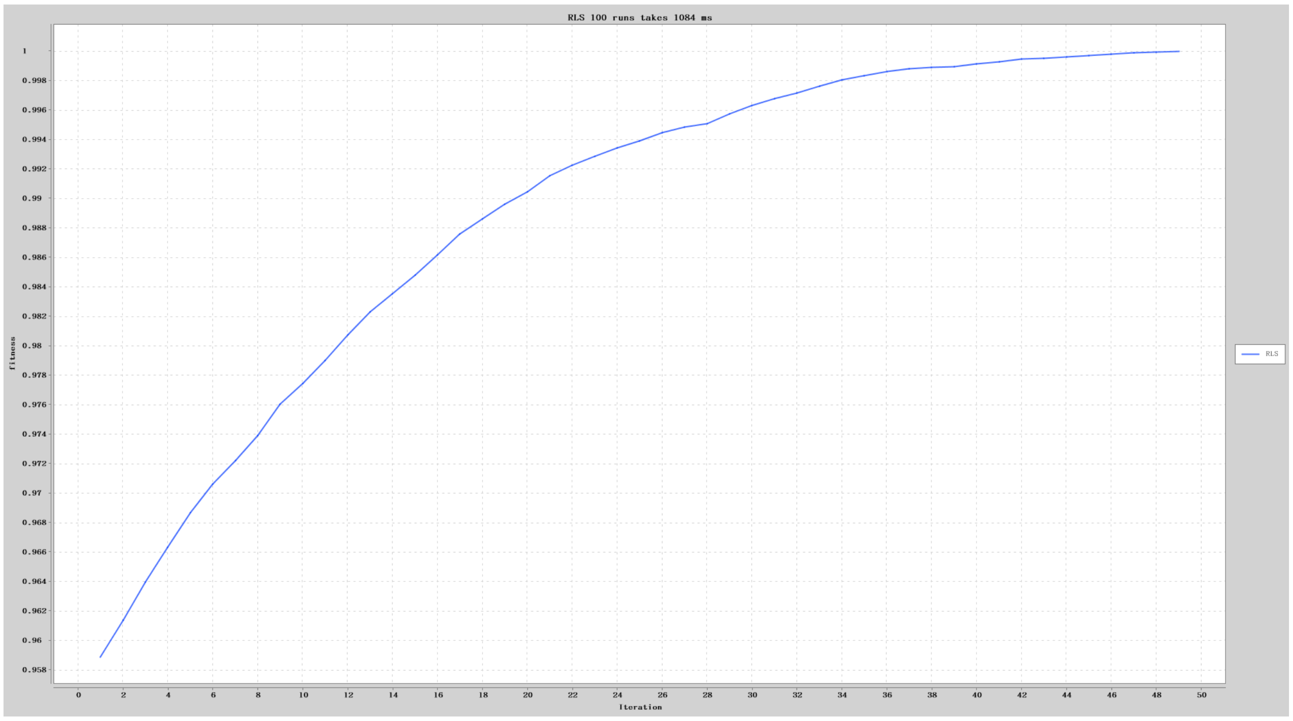


Figure A.25: RLS AllOnes

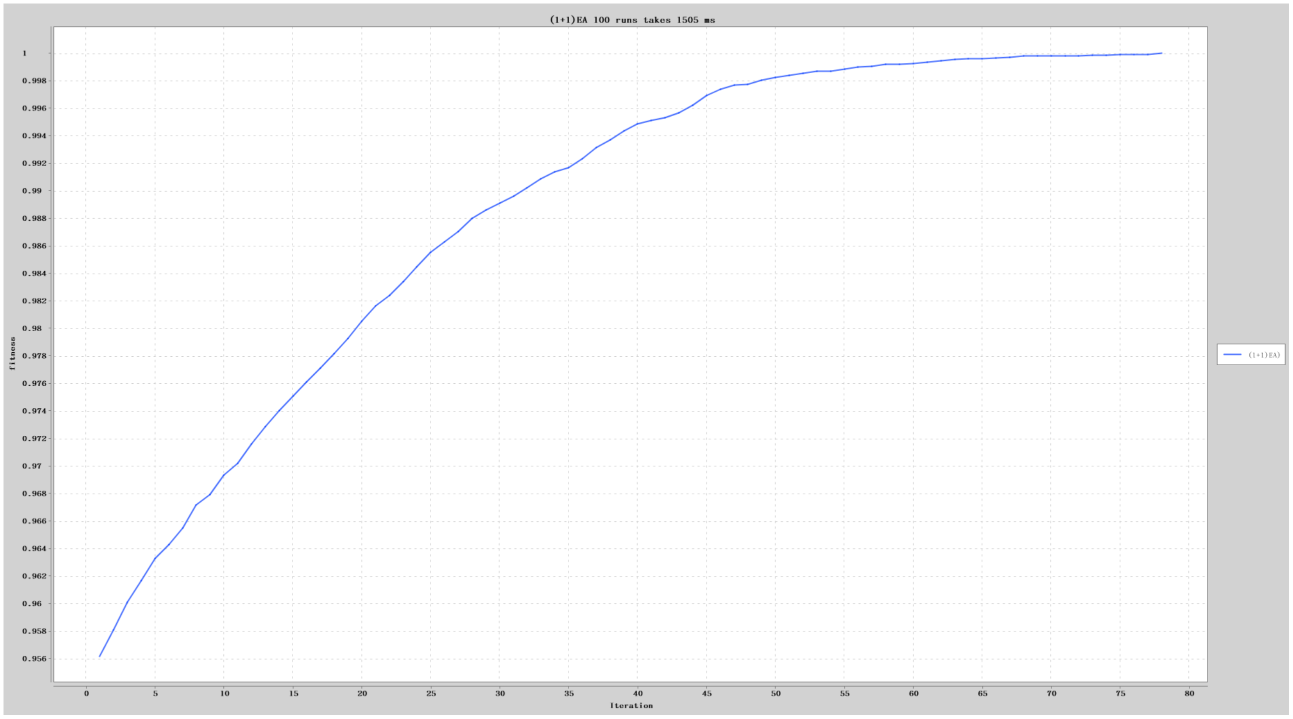


Figure A.26: (1+1)EA AllOnes

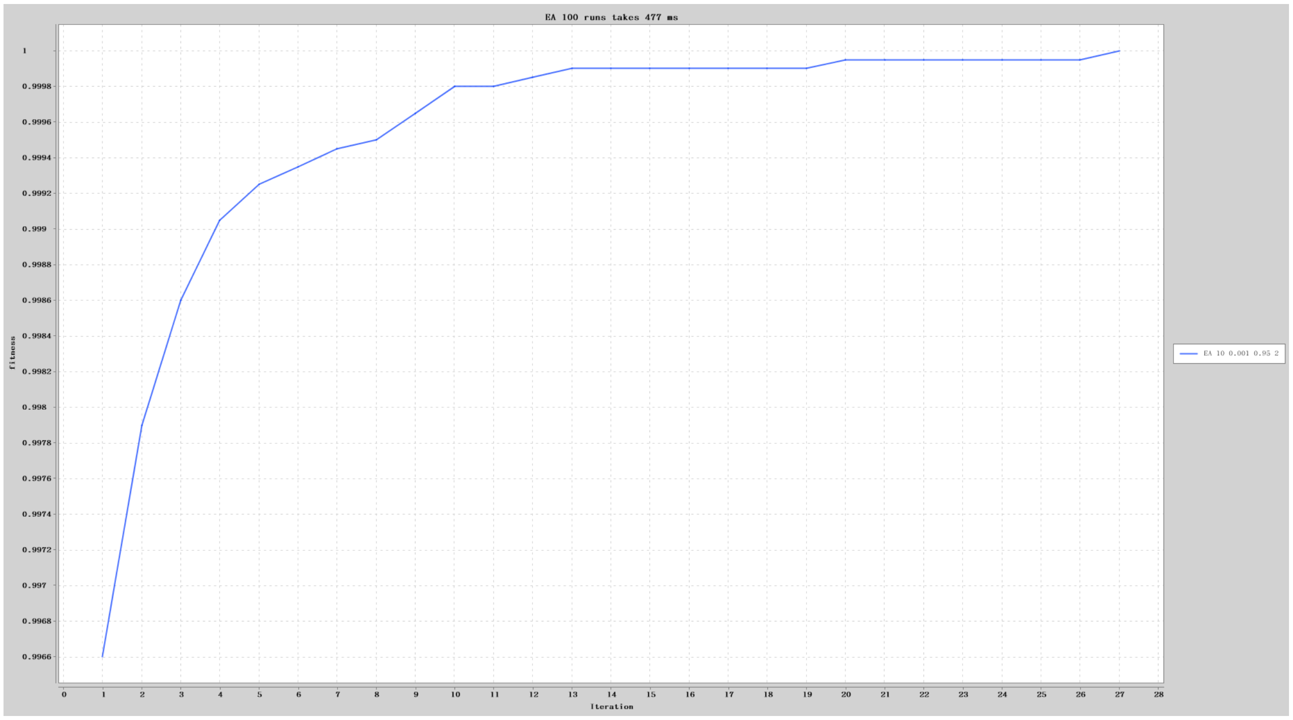


Figure A.27: EA AllOnes

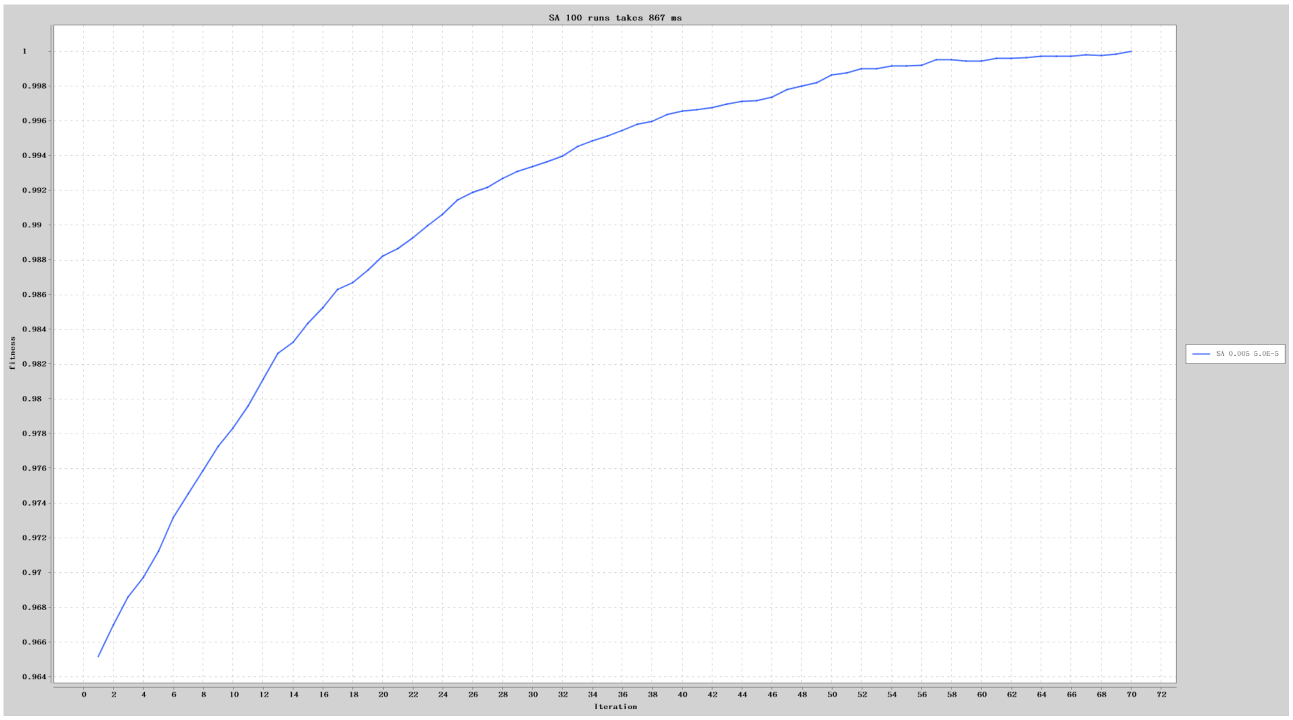


Figure A.28: SA AllOnes

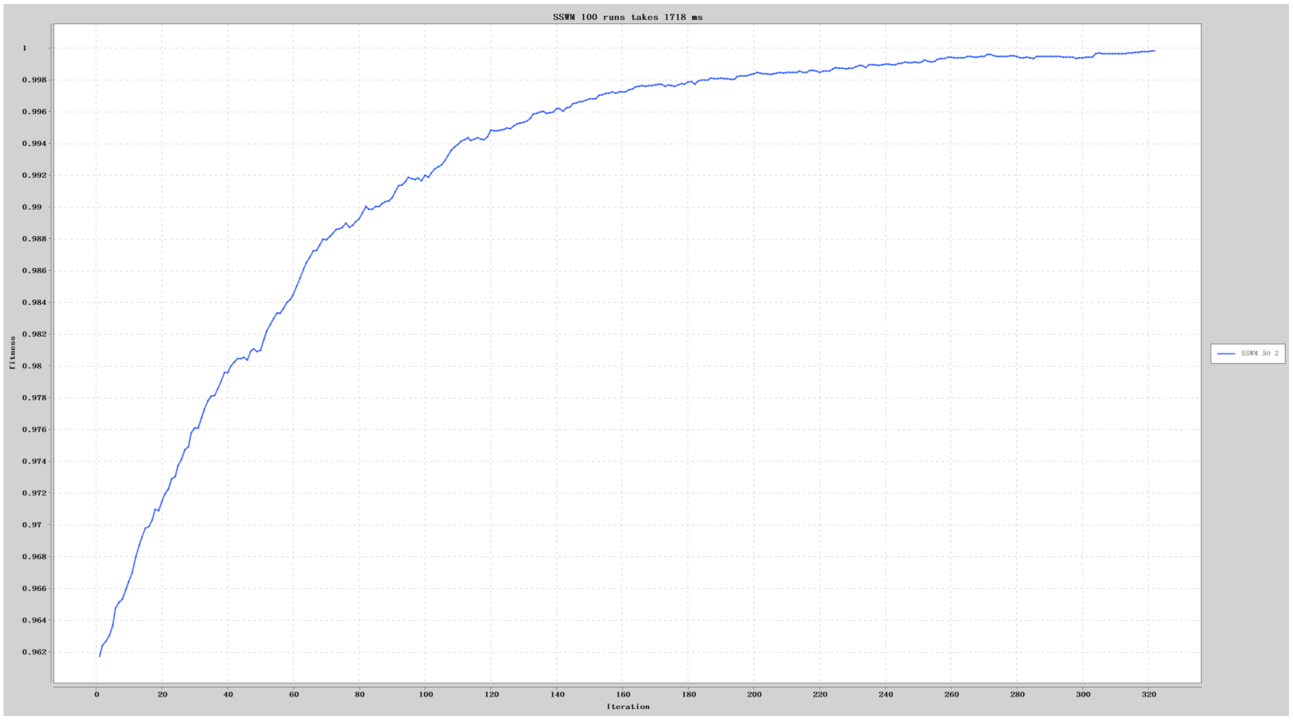


Figure A.29: SSWM AllOnes

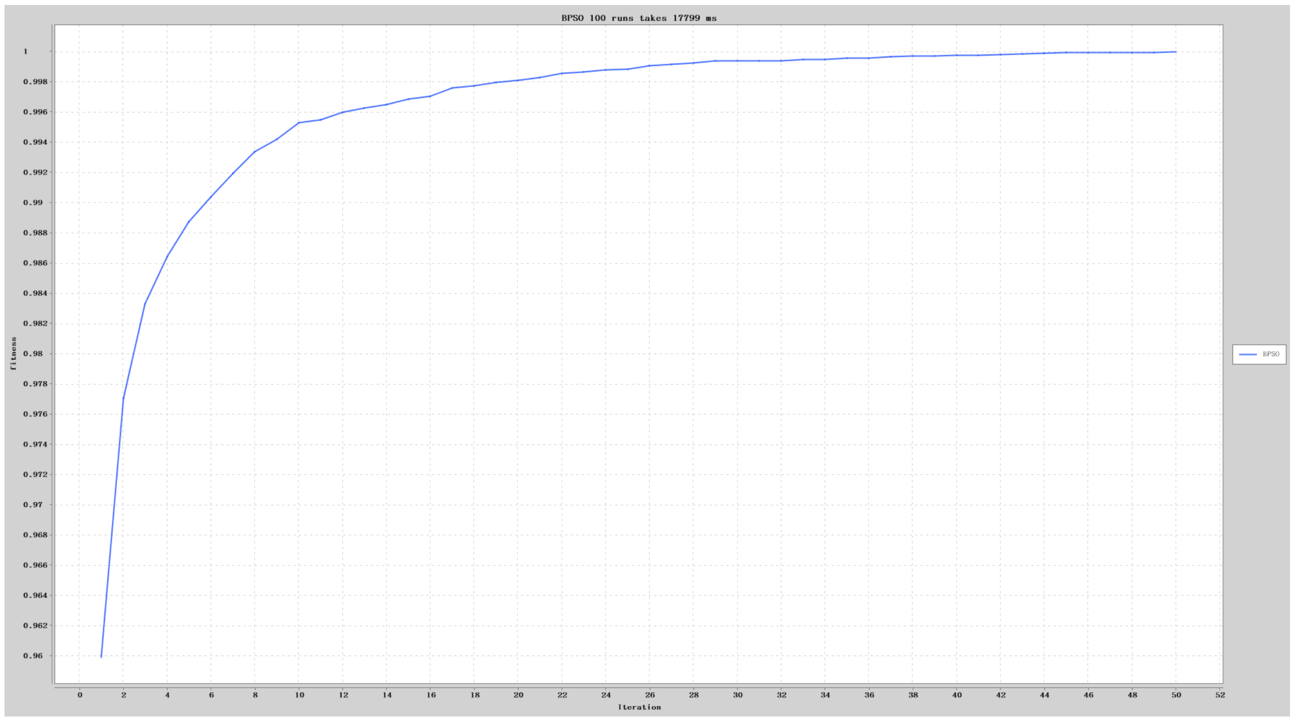


Figure A.30: BPSO AllOnes

A.3 Linearly Increased Jobshop

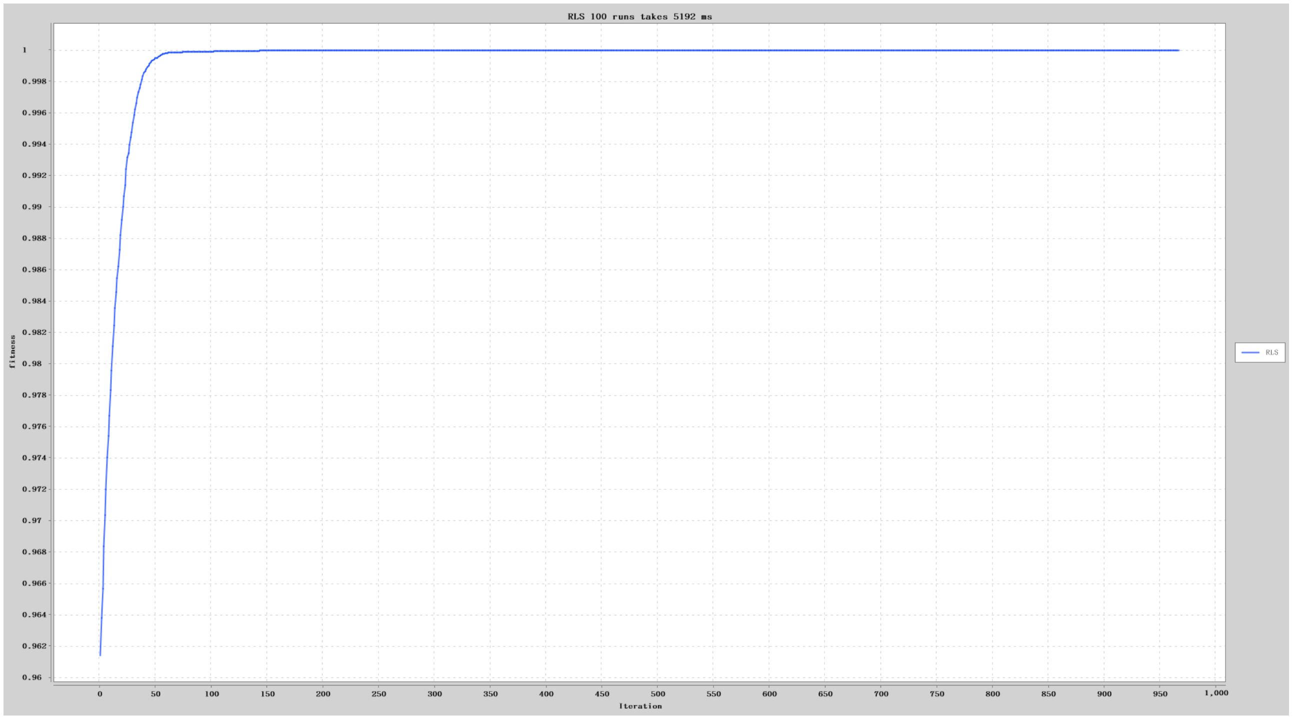


Figure A.31: RLS linear

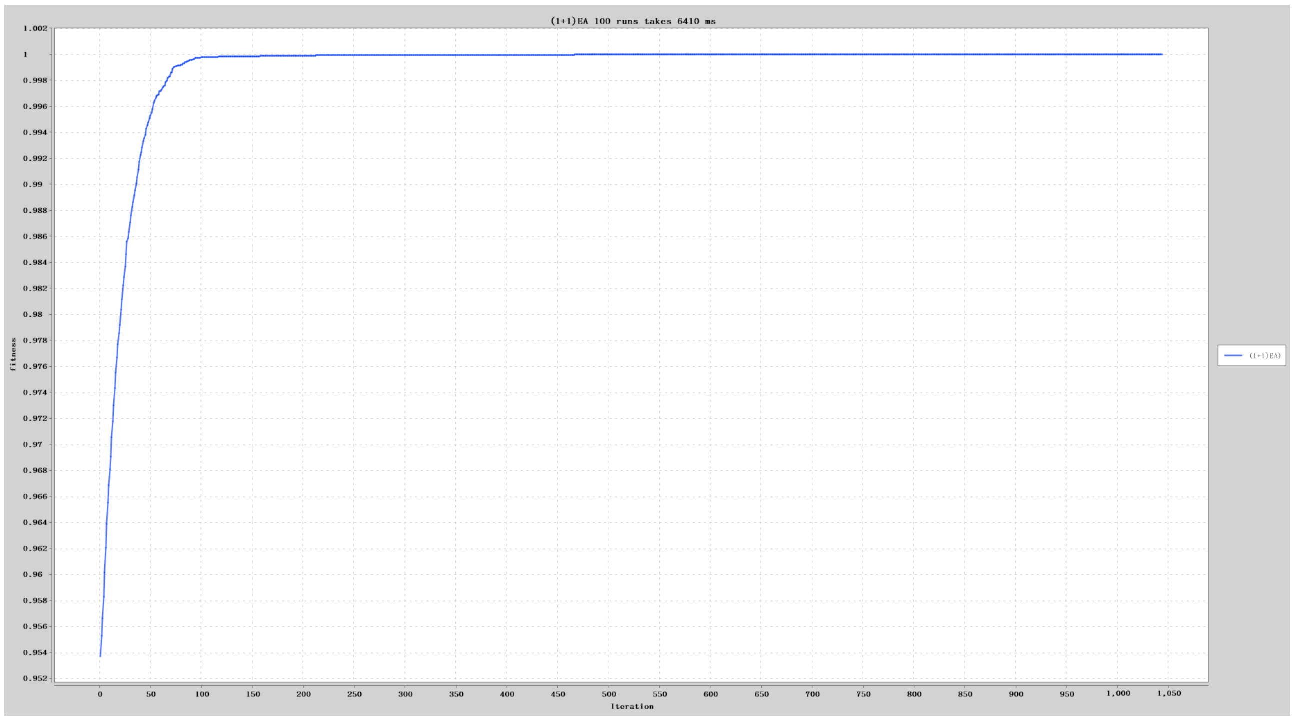


Figure A.32: (1+1)EA linear

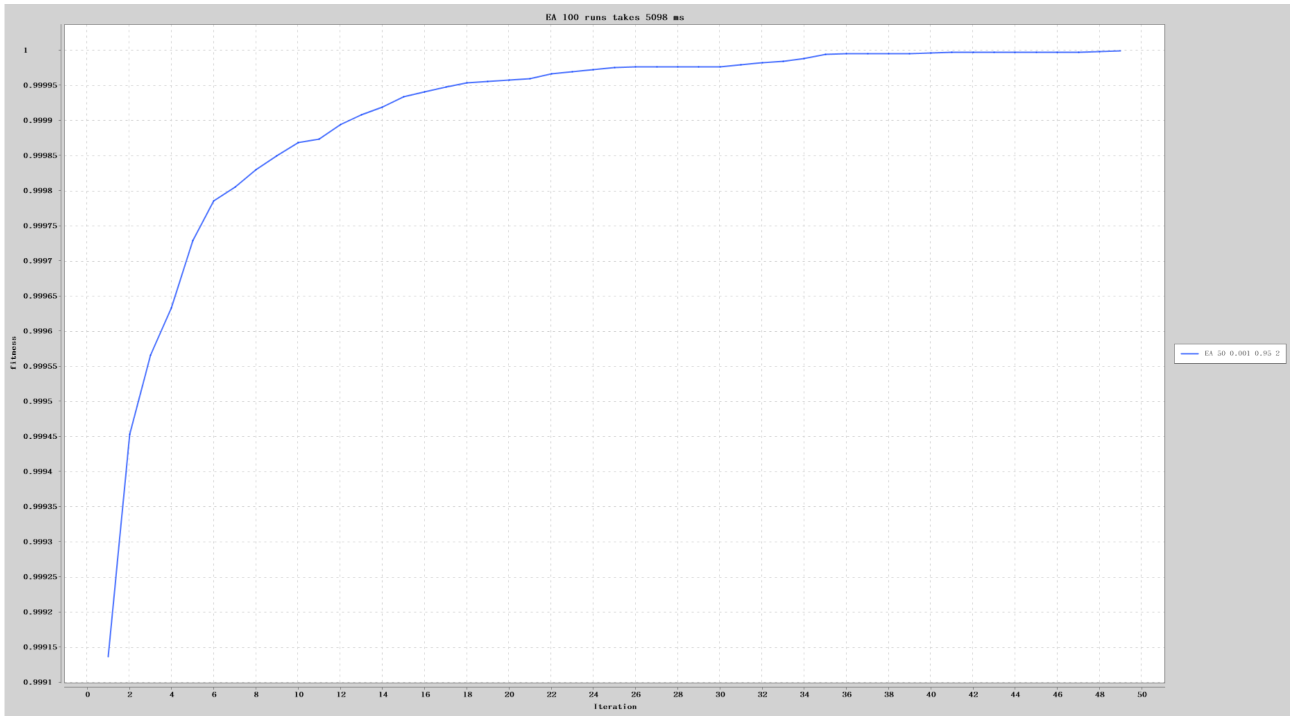


Figure A.33: EA linear

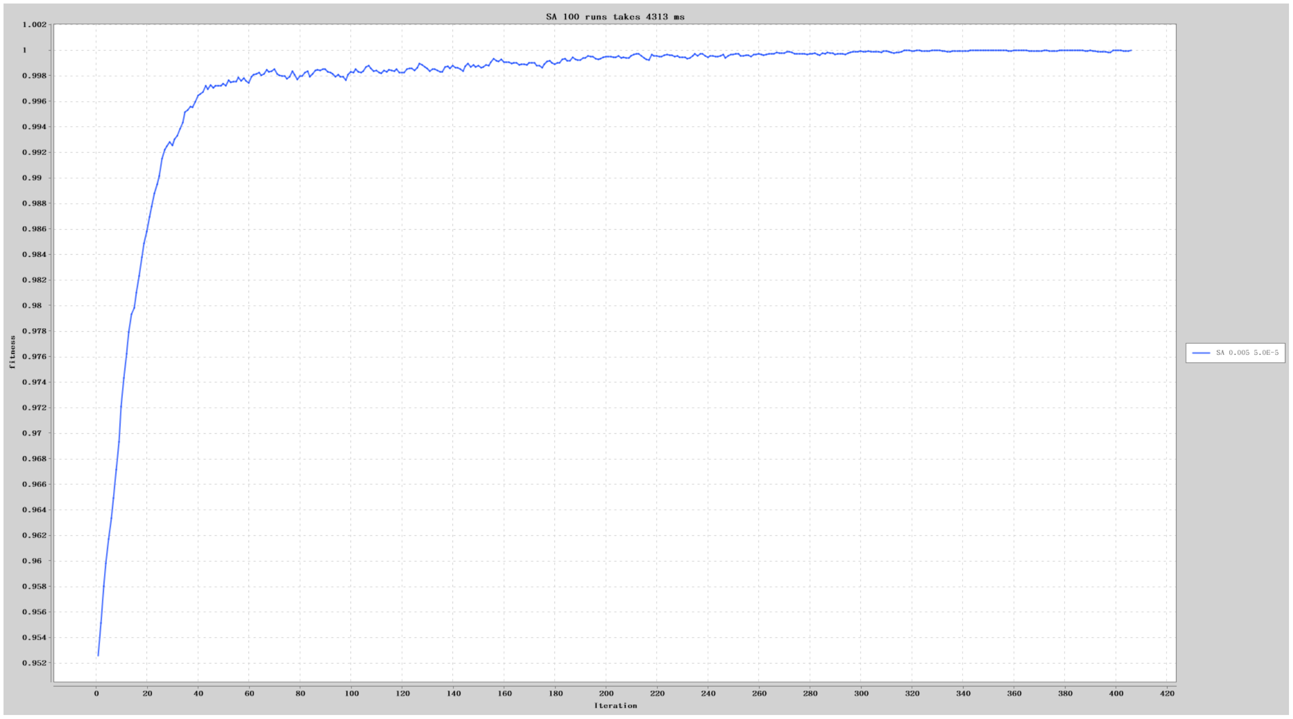


Figure A.34: SA linear

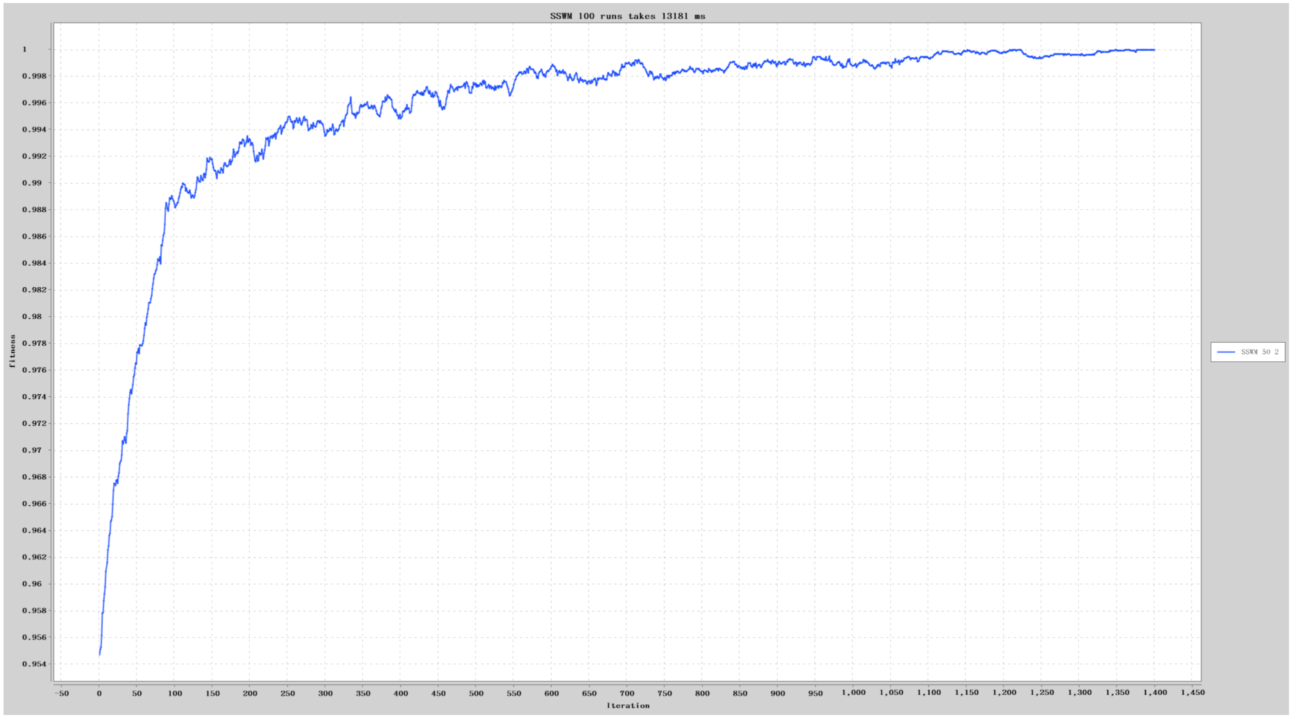


Figure A.35: SSWM linear

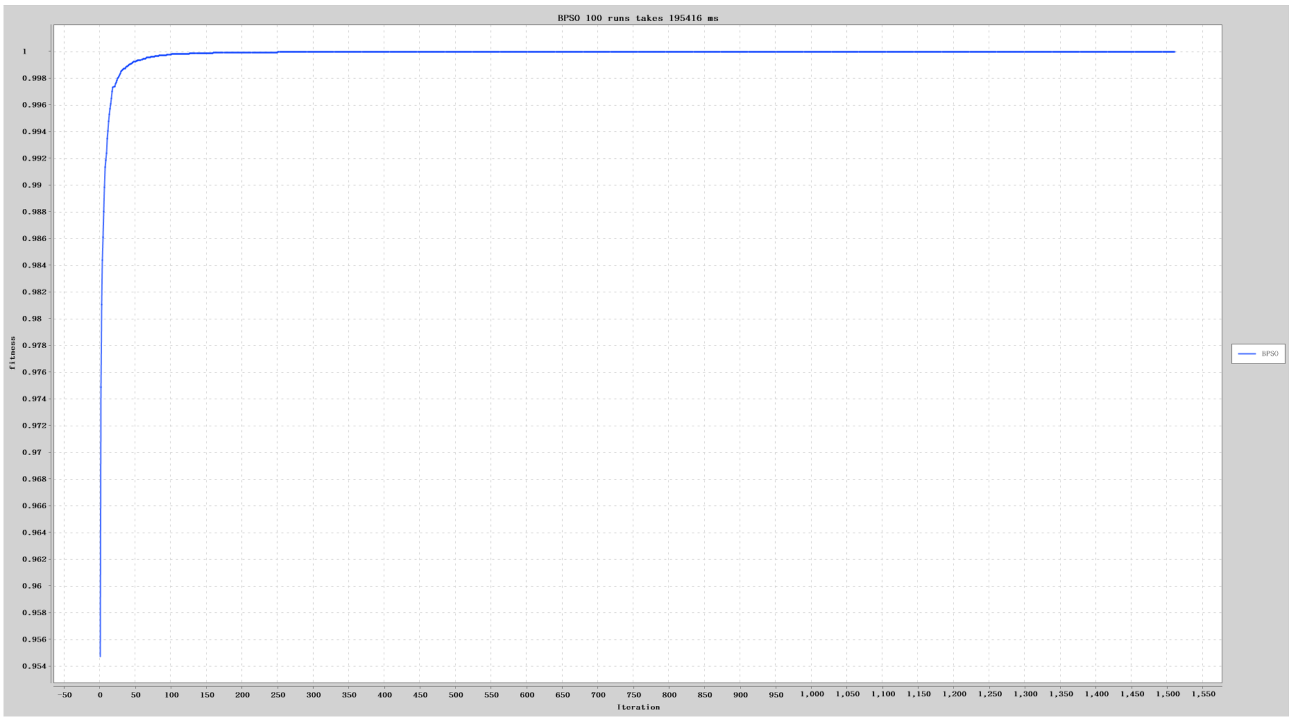


Figure A.36: BPSO linear

A.4 Worst Case Jobshop

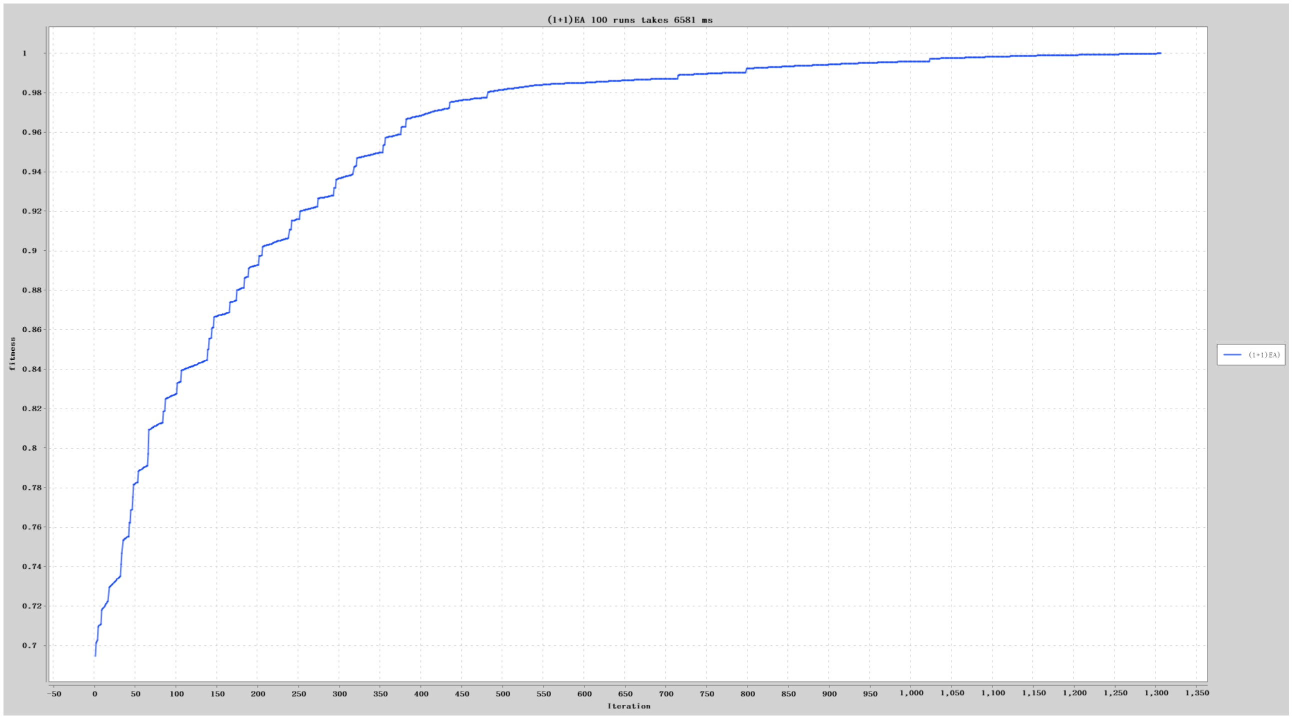


Figure A.37: (1+1)EA worst

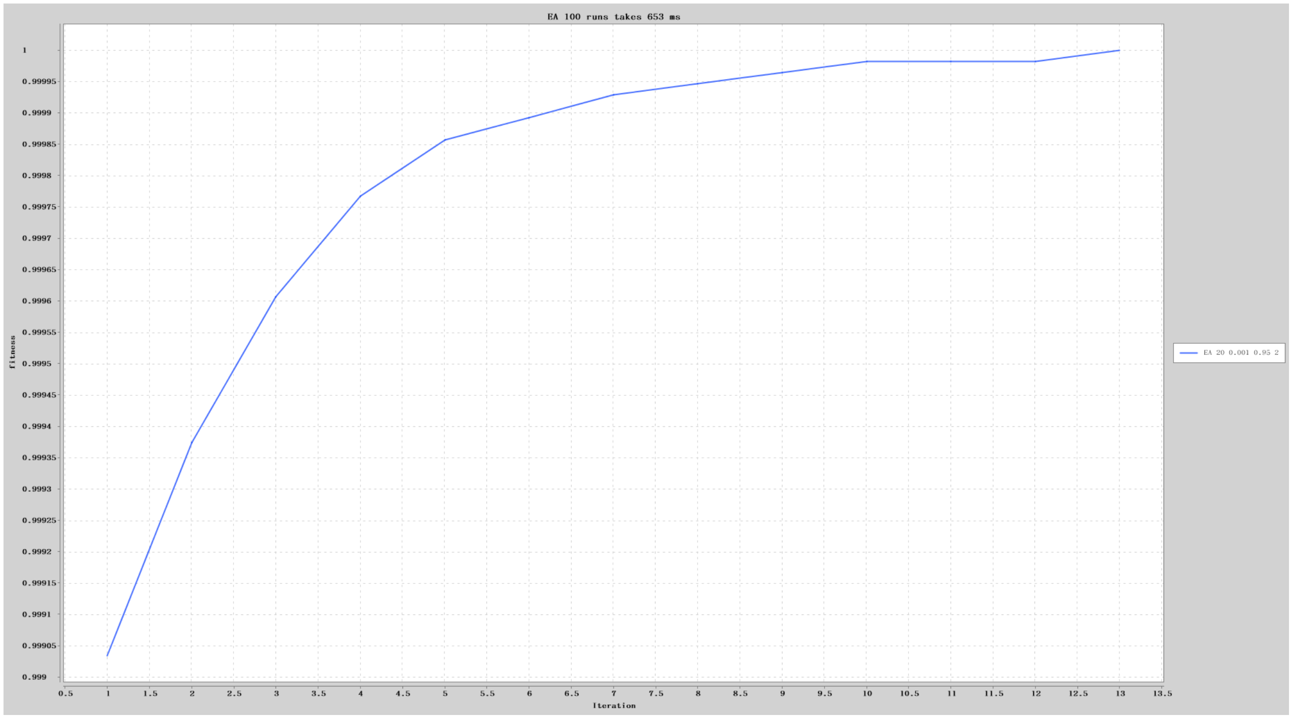


Figure A.38: EA worst

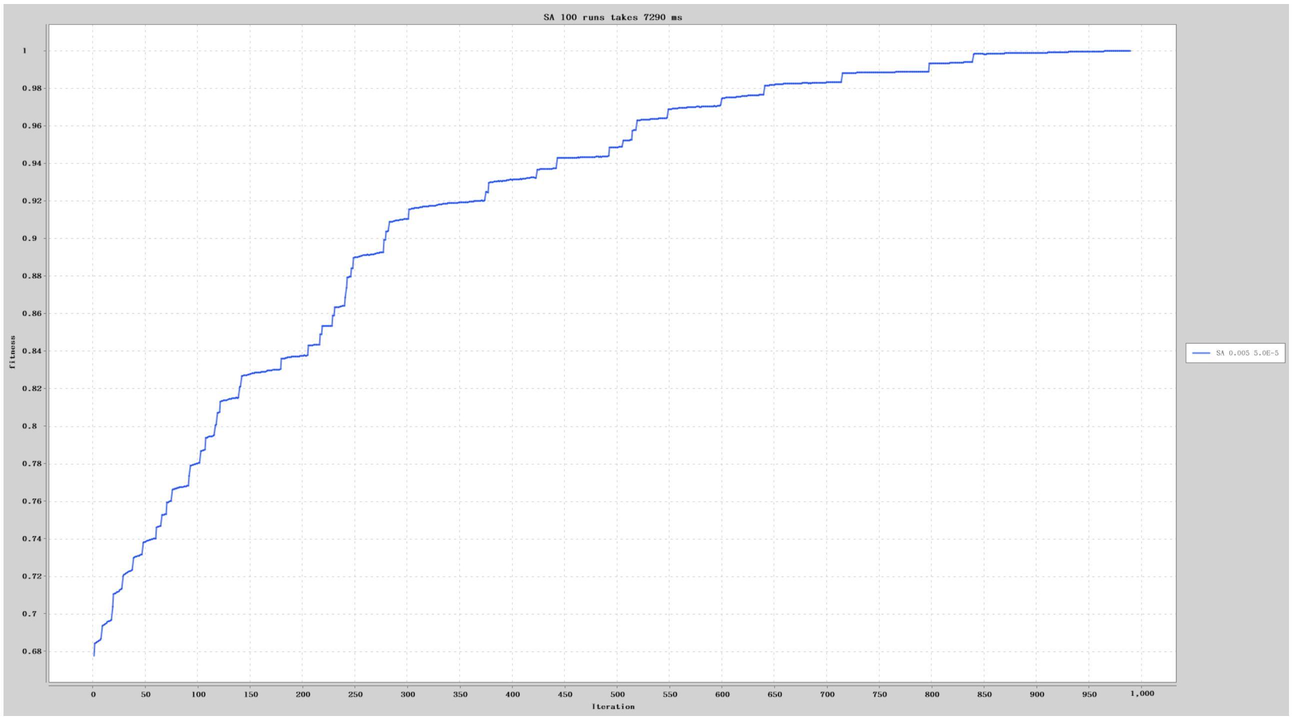


Figure A.39: SA worst

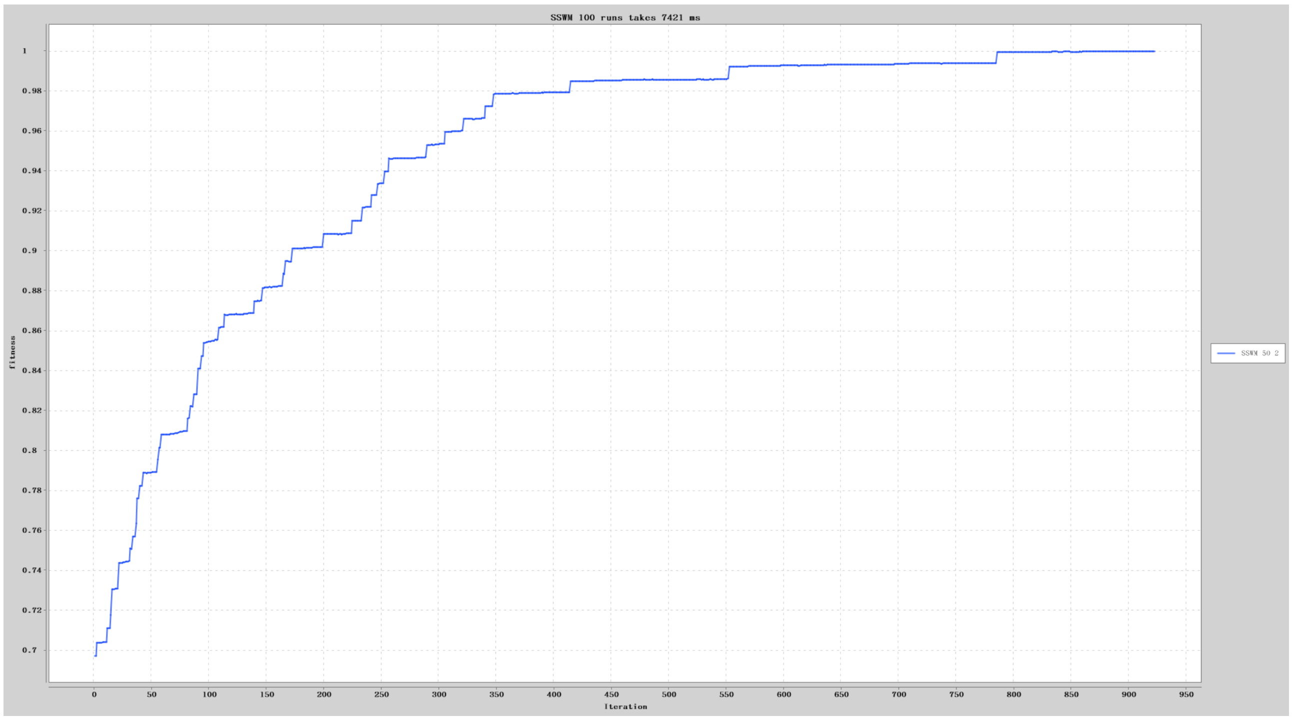


Figure A.40: SSWM worst

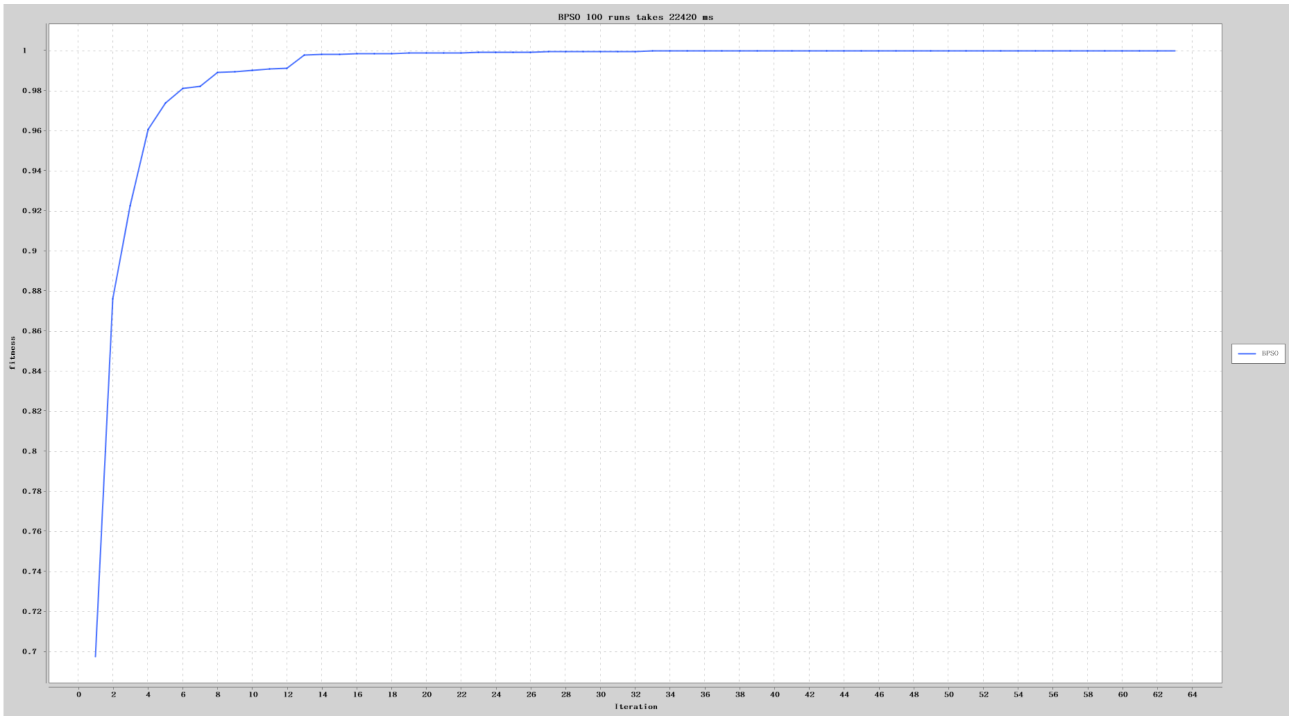


Figure A.41: BPSO worst