Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement.

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation.

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen.
The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.
Task Scheduling for FMS based on Genetic Algorithm

by

Hung-Yuan Li

Thesis submitted to the faculty of the Graduate School of the New Jersey Institute of Technology in partial fulfillment of the requirement for the degree of Master of Science in Electrical Engineering

1991
Title of Thesis: Task Scheduling for FMS Based on Genetic Algorithm

Name of Candidate: Hung-Yuan Li
Master of Science in Electrical Engineering

Thesis and Abstract Approved: ____________________________  __________
Dr. Edwin S. H. Hou, Advisor  Date
Assistant Professor
Department of Electrical & Computer Engineering

Signatures of other members of the thesis committee:

__________________________  __________
Dr. Anthony D. Robbi  Date
Associate Professor
Department of Electrical & Computer Engineering

__________________________
Dr. Nirwan Ansari  Date
Assistant Professor
Department of Electrical & Computer Engineering
VITA

Name: Hung-Yuan Li
364 Forest St. Apt. 3, Kerny, NJ 07032
(201) 991-1762

Education:

1989 - 1991 New Jersey Institute of Technology, M.S.E.E.
1982 - 1985 National Taipei Institute of Technology, B.S.E.E.

Position Held:

1985 - 1989 Tatung Co. Ltd., Electronic Engineer
ABSTRACT

Title of Thesis: Task Scheduling for FMS based on Genetic Algorithms
Hung-Yuan Li, Master of Science in Electrical Engineering, 1991
Thesis directed by: Dr. Edwin S. H. Hou
Department of Electrical and Computer Engineering

A Flexible Manufacturing System (FMS) consisting of p automated guided vehicles (AGV's), m workstations and n tasks is studied. The main problem investigated in this thesis is to find an optimal or suboptimal task scheduling for p AGV's among m workstations to complete n tasks.

An efficient approach based on genetic algorithms has been designed and implemented to solve the problem of task scheduling for a FMS. Near-optimal, or even optimal, task scheduling is accomplished by genetic algorithms. Simulation results on the algorithm are also discussed.
ACKNOWLEDGEMENT

I am genuinely grateful to Dr. Hou for his guidance and encouragement. In addition, I have also learned many practical and theoretical things, and felt this intractable thesis is almost enjoyable due to his help. I would like to thank the other professors in the thesis committee: Dr. Anthony D. Robbi and Dr. Nirwan Ansari for their advice and judgement. Thanks go to all professors and staffs who have taught and helped me in NJIT and let me have the potential capacity to complete this thesis. Last but not least, I would like to acknowledge Mr. Li-Bin Liu for his help on Microsoft Word for Windows while preparing this thesis.
Contents

1. Introduction ........................................ 1
   1.1 Flexible manufacturing system .................. 1
   1.2 Literature review ................................ 2
   1.3 Genetic algorithms .............................. 3
   1.4 Organization of thesis .......................... 4

2. Description of system model ....................... 5
   2.1 Problem formulation ............................ 5
   2.2 Assumptions of the system model .............. 6
   2.3 Representation of the system model .......... 9
   2.4 Cost function of a task schedule ............ 9

3. Description of genetic algorithms ................. 13
   3.1 Basic definitions of genetic algorithms ...... 13
   3.2 Genetic algorithms ............................ 14
   3.3 Examples of mutation operators .............. 15
4. Task scheduling in FMS using GA

4.1 Population size ............................................................ 17
4.2 Generating an initial population of schedules ..................... 17
4.3 Mutation operators and their selection ............................ 22
4.4 Reproduction ................................................................ 26
4.5 Complete algorithm ...................................................... 29

5. Simulation results ............................................................ 31

5.1 Task description ............................................................ 31
5.2 Tables for experiments .................................................. 35
5.3 Discussions .................................................................. 48

6. Conclusion ........................................................................ 50

References ........................................................................... 51

Appendix .............................................................................. 53

Software program listing (18 pages) ................................. 53
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>OR task</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>AND task</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>SINGLE task</td>
<td>8</td>
</tr>
<tr>
<td>5.1</td>
<td>T0 graph</td>
<td>32</td>
</tr>
<tr>
<td>5.2</td>
<td>T1 graph</td>
<td>32</td>
</tr>
<tr>
<td>5.3</td>
<td>T2 graph</td>
<td>32</td>
</tr>
<tr>
<td>5.4</td>
<td>Results of (T0,T1,T2)=(10,10,6) and 2 AGV's</td>
<td>44</td>
</tr>
<tr>
<td>5.5</td>
<td>Results of (T0,T1,T2)=(10,30,12) and 2 AGV's</td>
<td>44</td>
</tr>
<tr>
<td>5.6</td>
<td>Results of (T0,T1,T2)=(10,10,20) and 2 AGV's</td>
<td>45</td>
</tr>
<tr>
<td>5.7</td>
<td>Results of (T0,T1,T2)=(20,20,20) and 2 AGV's</td>
<td>45</td>
</tr>
<tr>
<td>5.8</td>
<td>Results of (T0,T1,T2)=(30,30,30) and 2 AGV's</td>
<td>46</td>
</tr>
<tr>
<td>5.9</td>
<td>Results of (T0,T1,T2)=(70,70,70) and 2 AGV's</td>
<td>46</td>
</tr>
<tr>
<td>5.10</td>
<td>Results of (T0,T1,T2)=(10,10,6) and 3 AGV's</td>
<td>47</td>
</tr>
<tr>
<td>5.11</td>
<td>Results of (T0,T1,T2)=(10,10,20) and 3 AGV's</td>
<td>47</td>
</tr>
</tbody>
</table>
Listing of Tables

5.1 The travelling time of AGV0 between workstations .................. 32
5.2 The travelling time of AGV1 between workstations .................. 32
5.3 The travelling time of AGV2 between workstations .................. 33
5.4 Table of (T0,T1,T2)=(10,10,6) and 2 AGV's ....................... 36
5.5 Table of (T0,T1,T2)=(10,30,12) and 2 AGV's ....................... 37
5.6 Table of (T0,T1,T2)=(10,10,20) and 2 AGV's ....................... 38
5.7 Table of (T0,T1,T2)=(20,20,20) and 2 AGV's ....................... 39
5.8 Table of (T0,T1,T2)=(30,30,30) and 2 AGV's ....................... 40
5.9 Table of (T0,T1,T2)=(70,70,70) and 2 AGV's ....................... 41
5.10 Table of (T0,T1,T2)=(10,10,6) and 3 AGV's ....................... 42
5.11 Table of (T0,T1,T2)=(10,10,20) and 3 AGV's ....................... 43
Chapter 1

Introduction

1.1 Flexible manufacturing system

A flexible manufacturing system (FMS) is a large and complex system typically consisting of a set of workstations; a material handling system (MHS) that connects these workstations by automated guided vehicles (AGV’s); and service centers (e.g., material warehouse, tool room, repair equipment). The workstation is an autonomous unit that performs certain manufacturing functions (e.g., a machining center, inspection machine, and a load-unload robot). The MHS is used for distributing the appropriate input to the workstations, so that the workstation can perform its tasks and remove from the workstation its output, e.g., ready products and worn tools [1], [2]. Typically, parts and materials in a FMS are efficiently and automatically conveyed via AGV’s between workstations for processing under computer control. To reduce cost and increase production, the planning and decision
control for a FMS includes balancing the workload of the workstations, task-order scheduling and dispatching, automated tool and material management.

1.2 Literature Review

Flexible manufacturing systems are being installed by many organizations in an effort to improve productivity. Because efficient operation of these systems is such a complex task, the concept of computer-based decision support systems promises to remedy the situation. After an FMS is built and configured, two main problems remain to be solved: planning and scheduling. These two problems can be formulated as the determination of an optimal task scheduling of P AGV's among M workstations to complete N tasks in an FMS. Various approaches regarding the planning and scheduling of FMS have been proposed by researchers [2]-[9]. P. E. Chen and J. Talavage [2] used a software package, Production Decision Support System, to assist the production decision maker in operating this complex manufacturing facility. R. Sui and C. K. Whitney [3] defined the structure of a decision support system to get the maximum benefit from an FMS. The structure of this decision support system parallels the organizational activities involved in running the FMS. A. Ballakur and H. J. Steudel [4] reviewed important theoretical and practical developments in job control. The distinguishing feature of this paper is the identification and summary of important concepts and procedures useful for incorporation into computerized job shop control system. Ho and Cao [5] used a perturbation analysis to estimate the sensitivity of system throughput with respect to routing probabilities in queuing networks and FMS. The use of mathematical programming, to establish the optimality of balanced workload for certain types of FMS's, was performed by Stecke and Morin [6]. The necessary planning and decision
control of an FMS includes balancing the workload of the workstations; work-order scheduling and dispatching; and automated tool and material management. These aspects of FMS have been discussed in [7] - [9]. In addition, C. L. Chen et al. [10], and P. S. Lui and L. C. Fu [11] proposed using A* search algorithm with minimax criterion and heuristic rules to solve the optimal task scheduling for FMS. A* search algorithm is a classical minimum-cost graph search method which guarantees finding an optimal solution by using heuristic information. The efficiency of this method is highly dependent on the heuristic information. In this thesis, we present an alternative approach which can efficiently find a solution based on genetic algorithms. The task scheduling problem in FMS can be thought as a generalization of the famous "Travelling Salesman Problem" which is known to be NP-complete.

1.3 Genetic algorithms

Genetic algorithms have been used to solve a wide variety of difficult and complicated optimization problems, such as, optimizations involving discontinuous, noisy, high-dimensional and multimodal objective function, combinatorial optimization [12], [13], and machine learning [14], [15]. The basic idea of genetic algorithms is based on mechanics of natural genetics and the notion of survival of the fittest. Typically, a genetic algorithm consists of the following four steps:

1) Initialize.
2) Evaluate fitness function.
3) Perform genetic operators.
4) Repeat step 2 and 3 until convergent.
1.4 Organization of thesis

This thesis is organized as follows: In chapter 2, we formulate the problem of task scheduling in FMS. Chapter 3 introduce genetic algorithms, and cite an example of a genetic operator. Chapter 4 explains how to find an optimal task scheduling by genetic algorithms. Chapter 5 describes the simulation results and chapter 6 concludes this thesis.
Chapter 2

Description of system model

2.1 Problem formulation

From the description in the previous chapter, the optimal routing assignment problem can be formulated by finding an optimal distribution of the appropriate tasks to the workstations by AGV's. To raise the productivity of an FMS, it is desirable to minimize the travelling time and job execution time of the AGV's among the workstations. The total finishing time (which includes job execution time and travelling time) of a routing assignment can be used as a cost function. Our objective naturally focuses on finding a task schedule with the shortest finishing time. In fact, the total finishing time of a routing assignment includes the total executing time (through workstations), the total travelling time and the total waiting time. Waiting occurs when two or more AGV's arrive at the same workstation, since only one AGV can work in a workstation.
2.2 Assumptions of the system model

We assume the AGV’s, workstations and the tasks have the following properties:

a ) An AGV will not take part in the execution of another task, until it has finished the present task assigned, that is, non-preemptive scheduling.

b ) There are no precedence and dependence constraints between tasks, but the internal subtasks of each task have constraints.

c ) Tasks are divided into 3 types: OR task, AND task, and SINGLE task.
   OR task: This type of task has branches so that each AGV has at least two different paths to select. An OR task indicates that subtasks of only one of the branches need to be accomplished. (See Fig. 2.1)
   AND task: This type of task has branches so that each AGV has at least two different paths to select. An AND task indicates that subtasks of all branches must be completed. There are no precedence and dependence relationships among these branches. (See Fig. 2.2)
   SINGLE task: This type of task has no branches, so each AGV has only one path to select. (See Fig. 2.3)

d ) The number of AGV’s, the number of tasks and the number of workstations are predetermined.

e ) All workstations are interconnected.
f) The location of every workstation is permanently fixed and predetermined. Each AGV has a different travelling speed, but its speed is fixed. Therefore, we can calculate the travelling time of any AGV from one workstation to the other workstation in advance.

g) All AGV's have the same job execution time in the same workstation. The launching position of the AGV's are not necessarily the same, because each AGV may start from any dispatching center according to different needs.

h) If there are two or more AGV's which want to enter the same workstation, we call this task collision and permit the AGV which arrives first, to enter the workstation. The other AGV's will wait until this AGV leaves the workstation. If two AGV's arrive simultaneously at the same workstation, then we assume AGV0 has the highest priority (the priority ordering of AGV's is AGV0, AGV1, ..., AGVp.).
T0 has two possible paths: T00 and T01
T00: w1 → w3 → w5 → w1
T01: w1 → w3 → w4 → w1

T1 has two possible paths: T10 and T11
T10: w1 → w7 → w3 → w6 → w1
T11: w1 → w3 → w7 → w6 → w1

T2 has only one path: T20
T20: w1 → w4 → w5 → w2
2.3 Representation of the system model

A FMS can be defined by a set $S=\{ A, W, T, N \}$, where $A$ denotes a set of AGV's, $W$ denotes a set of workstations, $T$ denotes a set of unique tasks and $N$ denotes the number of executions of each corresponding task. Namely, $A=\{ A_1, A_2, \ldots, A_p \}$, where $A_i$ indicates the $i$th AGV; $W=\{ W_1, W_2, \ldots, W_m \}$, where $W_i$ indicates workstation $i$; $T=\{ T_1, T_2, \ldots, T_k \}$, where $T_i$ indicates the $i$th type of task and $N=\{ N_1, N_2, \ldots, N_k \}$, where $N_i$ indicates the number of the $i$th type of task.

2.4 Cost function of a task schedule

Before we describe the cost function, we will first introduce the following definitions.

A ) Individual definitions:

a ) $ATT_{hk}$: Travelling time of an AGV between workstation $h$ and workstation $k$.

b ) $WET_h$: Executing time of workstation $h$.

c ) $AWT_i$: Waiting time of AGV $i$ for entering a workstation.

d ) $SFT_i$: Total finishing time of schedule $i$.

e ) $AFT_i$: The total finishing time of all tasks assigned to AGV $i$. 

To understand the following definitions, we suppose that there are k unique task type (T1, T2, ..., Tk) and the number of AGV's is p. We assume, for the special case, that every AGV is identically responsible for a sequence of tasks (T1,T2, ...,Tk).

B ) Integrated definitions:

a) Let TET ij represent the execution time of task i via AGV j (not including travelling time). Then the total execution time of a sequence of tasks (T1, T2, ..., Tk) spent by AGV j is defined as:

\[ ET_j = TET_{1j} + TET_{2j} + \ldots + TET_{kj} \]

Note. The execution time of any task is spent in workstations. The travelling time of any task is spent between workstations.

b) Let TT ij denote the travelling time of task i via AGV j, then the total travelling time of a sequence of tasks (T1, T2, ..., Tk) via AGV j is defined as:

\[ TT_j = TT_{1j} + TT_{2j} + \ldots + TT_{kj} \]

c) When AGVj and AGVi arrive at a workstation at the same time, or the workstation is executing a subtask for AGVj when AGVi arrives at the workstation, then AGV i has to wait until the workstation has completed work and AGV j has left as well. The total waiting time for AGV j in a sequence of tasks (T1, T2, ..., Tk) is defined as:

\[ WT_j = WT_{1j} + WT_{2j} + \ldots + WT_{kj} \]
From the above definitions, the total finishing time of $T_1$ through $T_k$ taken by AGV $j$ is defined as:

$$TA_j = ET_j + TT_j + WT_j$$

Suppose we are searching $v$ schedules where every schedule has $p$ AGV's and every AGV processes the sequence of tasks ($T_1, T_2, \ldots, T_k$). Then the total finishing time of the $i$th schedule ($i$ is between 1 and $v$) is defined as:

$$SFT_i = \max \{ TA_1, TA_2, \ldots, TAp \}$$

And, the total finishing time of the best routing assignment among the $v$ schedules is defined as:

$$BSFT = \min \{ SFT_1, SFT_2, \ldots, SFTv \}$$

Let us illustrate the above definitions with the following example. Consider a schedule with 2 AGV's and 2 tasks.

<table>
<thead>
<tr>
<th>Schedule</th>
<th>Tasks</th>
<th>AGV0</th>
<th>AGV1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_0$</td>
<td>w1----&gt;w4----&gt;w6----&gt;w2</td>
<td>3 4 6 3 7 6 3</td>
<td>2 3 7 4 9 3 2</td>
</tr>
<tr>
<td>$T_1$</td>
<td>w0----&gt;w3----&gt;w5----&gt;w2</td>
<td>e te te te te te</td>
<td>e te te te te te</td>
</tr>
</tbody>
</table>
Let us calculate the total finishing time of the schedule by using the following and data:

<table>
<thead>
<tr>
<th>Starting point of AGV's</th>
<th>Ending point of AGV0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 3 7 13 16 23 29 30 33</td>
<td></td>
</tr>
</tbody>
</table>

AGV0 | w1 | tr | w4 | tr | w6 | tr | col | w2 |
AGV1 | w0 | tr | w3 | tr | w5 | tr | w2 |

0 2 5 12 16 25 28 30 (time unit)

Ending point of AGV1

col: collision happened e : execution time

tr : AGV is travelling t : travelling time

wi : ith workstation

The results are

$$\begin{align*}
ET0 &= 3 + 6 + 7 + 3 = 19; \quad ET1 = 2 + 7 + 9 + 2 = 20 \\
TT0 &= 4 + 3 + 6 = 13; \quad TT1 = 3 + 4 + 3 = 10 \\
WT0 &= 1; \quad WT1 = 0 \\
TA0 &= ET0 + TT0 + WT0 = 33; \quad TA1 = ET1 + TT1 + WT1 = 30
\end{align*}$$

So, $SFT = \max \{ TA0, TA1 \} = 33$ (time unit); $BSFT = 33$.

Note. $T0$ has a collision and $T1$ does not.
Chapter 3

Description of genetic algorithms

3.1 Basic definitions of genetic algorithms

a) Gene: A gene is the smallest element.
b) Chromosome: A chromosome includes a set of genes.
c) String: A string denotes the parameter of the search space and consists of a set of chromosomes.
d) Population: A population consists of a set of strings.
e) Mutation: A process of changing the value of a gene.
f) Crossover: The process of generating a new string by joining portions of two old strings.
g) Reproduction: The process of selecting new strings from an old population of strings based on their fitness values.
3.2 Genetic algorithms

Genetic algorithms have been successfully applied to various optimization problems, such as, travelling salesman problem, gas pipeline optimization, etc. The success of genetic algorithms can be attributed to the following principles:

1) Genetic algorithms use a coding of the parameter set rather than the parameters themselves.
2) Genetic algorithms search from a population of search nodes instead of a single one.
3) Genetic algorithms use probabilistic transition rules.

A genetic algorithm consists of a string representation ("genes") of the nodes in the search space, a set of genetic operators for generating new search nodes, a fitness function to evaluate the search nodes, and a stochastic assignment to control the genetic operators.

The concise steps are summarized as follows:

a) Initialization: An initial population of strings are constructed at random.

b) Evaluation of fitness function (cost function): The fitness value of each string is calculated to allow us to judge whether the string is good or bad.
c) Application of genetic operators: After evaluating the fitness value of each string, we can apply the designed genetic operators to the old population to generate a new population.

d) Repeat (b) and (c) until convergent ("convergent" means that there is no better solution than the latest solution.).

The application of genetic algorithms is controlled by a set of stochastic assignments. These stochastic assignments together with the design of the genetic operators will greatly affect the performance of the algorithm and the results obtained. From the above description, we can see that genetic algorithms utilize the notion of survival of the fittest; passing "good" genes to the next generation of strings, and combining different strings to explore new search points.

3.3 Examples of mutation operators

In general, the mutation operator works by changing the value of a randomly selected gene or by exchanging the value of two genes. For example,

a) When the strings are represented as binary strings, mutation can be implemented by first choosing a bit at random. If the bit is 1 (0), then we replace it with 0 (1). For instance, old string = 100001010*, the new string generated after mutation will be = 100011010.

* The bit position is selected at random.
b) For this mutation operator, we first randomly select two characters and then exchange these two characters. For example:

Old string: A B C D E F G H I J
New string: A B H D E F G C I J

In GA’s, mutation serves the crucial role of replacing the genes lost from the population during the selection process, so that they can be tried in a new context, or providing the genes that were not present in the initial population.
Chapter 4

Task scheduling in FMS using GA

4.1 Population size

The optimal number of strings in a population is largely determined by experiment. In fact, if we select a large population size, the genetic algorithm will spend much more time to run and may not find the best solution. At the same time, if the population size chosen is too small, the optimal routing assignment might be missed when GA prematurely converges. Based on our experience, a population size of 20 is used.

4.2 Generating an initial population of schedules

Suppose there are $n$ type of tasks ($T_1 + T_2 + \ldots + T_n$), $p$ AGV's and $m$ workstations in a FMS. For convenience, we temporarily do not consider the
travelling time between any two sequential tasks and the waiting time due to task collisions. If the finishing time of each task is defined as TFT_i (i is between 1 and n), then the total finishing time for the n tasks is TFT where TFT = TFT_1 + TFT_2 + ... + TFT_n. Since there are p AGV’s in the FMS, every AGV basically works n/p tasks and spends n/p TFT. We use an example here to clarify these relationships:

Given: n = 5 : T1, T2, T3, T4, T5 ; 2 AGV’s : AGV1, AGV2

TFT1 = 30, TFT2 = 20, TFT3 = 12, TFT4 = 10, TFT5 = 8

TFT = TFT_1 + TFT_2 + TFT_3 + TFT_4 + TFT_5

= 30 + 20 + 12 + 10 + 8 = 80

So, AFT1 = AFT2 = 80 / 2 = 40

AFT_j : The total finishing time of tasks assigned to AGV_j

To roughly distribute the task load evenly, first we randomly select a task (from n tasks as the first task to be executed via AGV1. If AFT1 < 1/2 TFT (=40), then we continue to randomly add tasks until the total finishing time via AGV1 is larger than 1/2 TFT. The rest of the n tasks, which were not selected by AGV1, will be all executed by AGV2. Although this method generates initial task arrangements, it can be modified to produce better task arrangements. For example,

AGV1 : T1 --> T2 (AFT1 = 30 + 20 = 50)

AGV2 : T3 --> T4 --> T5 (AFT2 = 12 + 10 + 8 = 30)

The difference between the total finishing time of the two AGV’s is 50 - 30 = 20. This schedule is not a good routing assignment, because the workload is not balanced in the two AGV’s. We can utilize another method to modify this assignment. The method is to balance the workload for all the AGV’s, especially if the maximum
difference of the finishing times for any two tasks is large. We called the first method "coarse arrangement", and the second method is called "fine arrangement". The steps for fine arrangement is described in the following:

0) Initialization 1: loop=0; u=1
1) Initialization 2: i=1 and j=1; Supposed AGV L (0 < L < p+1) has the longest total finishing time (= LAFT) in the uth schedule and AGV S (0 < S < p+1) is assumed with the shortest total finishing time (= SAFT) in the uth schedule (0 < u < v+1). Assume AGV L handles nl tasks and AGV S handles ns tasks (nl+ns=n).

Note. Every schedule includes P AGV's which must start from the same time but do not necessarily end at the same time. The total finishing time of this schedule is equal to that of the AGV with longest total finishing time in all AGV's.

2) Select the ith task (0 < i < m+1) in AGV L and select the jth task (0 < j < n+1) in AGV S.
3) Exchange the previous two tasks.
4) If | LAFT - SAFT | > | NLAFT - NSAFT |, then we keep the new task ordering and set LAFT=NLAFT and SAFT=NSAFT. Otherwise, restore the original task ordering. If | NLAFT - NSAFT | = 0, then stop.
5) i=i+1; If i = m+1, then go to step 6. Otherwise go to the above step 2.
6) Initialization 3: i=1 and j=1; Continue to use AGV L, AGV S, LAFT and SAFT in step 5.
7) Select the ith task (i is between 1 and m) in the AGV L and select the jth task (j is between 1 and n) in the AGV S.
8) Exchange the previous two tasks.
9) If $|\text{LAFT} - \text{SAFT}| > |\text{NLAFT} - \text{NSAFT}|$, then we keep the new task ordering and set $\text{LAFT} = \text{NLAFT}$ and $\text{SAFT} = \text{NSAFT}$. Otherwise, restore the original task ordering. If $|\text{NLAFT} - \text{NSAFT}| = 0$, then stop.

10) $j = j + 1$; If $j = n + 1$, then go to step 11. Otherwise go to the above step 7.

11) Perform this refinement process for five times. $\text{loop} = \text{loop} + 1$; If $\text{loop} = 5$, then go to step 12. Otherwise go to step 1.

12) $u = u + 1$; If $u = v$, then stop. Otherwise $\text{loop} = 0$ and go to step 1.

The above processing steps are slightly complicated, so we will use an example to illustrate the above procedure,

Ex:

$\text{AGV1}: T1 \rightarrow T2 \ (AFT1=30+20=50)$

$\text{AGV2}: T3 \rightarrow T4 \rightarrow T5 \ (AFT2=12+10+8=30)$

$n_1=2$, $ns=3$

Step 1. $\text{AGV L}=\text{AGV1}$; $\text{AGV S}=\text{AGV2}$; $\text{LAFT}=AFT1$ and $\text{SAFT}=AFT2$.

Step2. $T1$ is the $i$th ($i=1$) task in $\text{AGV1}$; $T3$ is the $j$th ($j=1$) task in $\text{AGV2}$.

Step3. Exchange the two tasks in step 2, so the new task arrangement becomes the following:

$\text{AGV1}: T3 \rightarrow T2 \ (NLAFT=12+20=32)$

$\text{AGV2}: T1 \rightarrow T4 \rightarrow T5 \ (NSAFT=30+10+8 = 48)$

$\text{LAFT} - \text{SAFT} = 50 - 30 = 20$; $\text{NSAFT} - \text{NLAFT} = 48 - 32 = 16$
Step 4. Since $|\text{LAFT} - \text{SAFT}| > |\text{NLAFT} - \text{NSAFT}|$, we accept the new arrangement and $\text{LAFT}=48$ and $\text{SAFT}=32$.

Step 5. $i=i+1$; Since $i (=2)$ is not equal to $m+1 (=3)$, go to step 2 below.

Step 2. $T_2$ is the $i$th ($i=2$) task in the AGV1; $T_1$ is the $j$th ($j=1$) task in the AGV2.

Step 3. Exchange the two tasks in step 2, so the new task arrangement becomes the following:

AGV1: $T_3$ $\rightarrow$ $T_1$ (NLAFT$=12+30=42$)
AGV2: $T_2$ $\rightarrow$ $T_4$ $\rightarrow$ $T_5$ (NSAFT$=20+10+8=38$)

$\text{LAFT}-\text{SAFT}=48-32=16$; $\text{NSAFT}-\text{NLAFT}=42-38=4$

Step 4. Since $|\text{LAFT} - \text{SAFT}| > |\text{NLAFT} - \text{NSAFT}|$, $\text{LAFT}=42$ and $\text{SAFT}=38$.

Step 5. $i=i+1$; Since $i (=3)$ is equal to $m+1 (=3)$, go to step 6.

Step 6. $i=1$, $j=1$, AGV1 = AGV1, AGV2 = AGV2, LAFT = 42, SAFT = 38 and the task ordering is as follows:

AGV1: $T_3$ $\rightarrow$ $T_1$ (NLAFT$=12+30=42$)
AGV2: $T_2$ $\rightarrow$ $T_4$ $\rightarrow$ $T_5$ (NSAFT$=20+10+8=38$)

Step 7. $T_3$ is the $i$th ($i=1$) task in the AGV1; $T_2$ is the $j$th ($j=1$) task in the AGV2.

Step 8. Exchange the two tasks in step 7, so the new task arrangement becomes the following:

AGV1: $T_2$ $\rightarrow$ $T_1$ (NLAFT$=20+30=50$)
AGV2: $T_3$ $\rightarrow$ $T_4$ $\rightarrow$ $T_5$ (NSAFT$=12+10+8=30$)

$|\text{LAFT}-\text{SAFT}| = 42 - 38 = 4$; $|\text{NSAFT}-\text{NLAFT}| = 50 - 30 = 20$

Step 9. Since $|\text{LAFT} - \text{SAFT}| < |\text{NLAFT} - \text{NSAFT}|$, $\text{LAFT}=42$ and $\text{SAFT}=38$.

We do not accept the arrangement and the task ordering is as follows:
AGV1: T3-->T1 (NLAFT=12+30=42)
AGV2: T2-->T4-->T5 (NSAFT=20+10+8 = 38)

Step10. j=j+1; Since j (=2) is not equal to n+1 (= 4), go to step7 below.

Step7. T3 is the ith (i=1) task in the AGV1; T4 is the jth (j=2) task in the AGV2.

Step8. Exchange the two tasks in step7, so the new task arrangement becomes the following:
AGV1: T4-->T1 (NLAFT=10+30=40)
AGV2: T2-->T3-->T5 (NSAFT=20+12+8 = 40)
| LAFT-SAFT | = 42 - 38 = 4; | NSAFT - NLAFT | = 40 - 40 = 0

Since NLAFT=NSAFT=40, this means that the workload is quite good for every AGV. We will use this task ordering as an initial schedule and stop here.

By interchanging the tasks between AGV's under any schedule, fine arrangement can generate better task arrangements. The resulting task arrangements will allow genetic algorithms to find the solution more quickly.

4.3 Mutation operators and their selection

Here, we propose 7 types of mutation operators described in the following:

1 ) Mutation1 : We randomly select two colliding tasks from two different AGV's in the same schedule, and exchange them.
Ex: Old schedule: (2 AGV's)

AGV1: T11, T24, T00  T11 and T24 are collided.
AGV2: T22, T10, T11  T10 and T11 are collided.

Note. Tasks with double underline have collisions.

The colliding tasks in AGV1 and AGV2 are respectively put in sets C1 = \{ T11, T24 \} and C2 = \{ T10, T11 \}. We randomly select two colliding tasks, one from each of C1 and C2 and exchange them. By the above selection policy, the new schedule becomes:

New schedule:

AGV1: T11, T10, T10

AGV2: T22, T24, T11  * T10 and T24 are selected

2) Mutation 2: Randomly select two colliding tasks from the same AGV, and exchange them. The selection of the two colliding tasks is the same as that of mutation 1. First, we put all the colliding tasks for the same AGV in a set, then randomly choose two different colliding tasks from the set.

Ex: Old schedule:

AGV1: T11, T22, T00
AGV2: T24, T10, T00

New schedule:

AGV1: T22, T11, T00
AGV2: T24, T10, T00
3) Mutation3: Assuming that T0 has two possible paths, T00 and T01, T1 also
has two possible paths: T10 and T11, and T2 has 6 possible paths: T20, T21,
T22, T23, T24 and T25. Randomly choose one colliding task from an AGV
and exchange it with another randomly selected path which has the same type
of task. For example, T00 can be replaced by T01 or vice versa. However, we
cannot exchange T00 and T24, since this will generate incorrect number of
tasks in the schedule.

Ex: Old schedule:

AGV1 : T11, T22, T00
AGV2 : T24, T10, T11

New schedule:

AGV1 : T11, T23, T00
AGV2 : T21, T10, T11

4) Mutation4 : Randomly choose a colliding task and a normal task from the
same AGV and exchange them (a normal task is a task that does not collide.).
The selection of the normal (or colliding) task is the same as that of
mutation1.

Ex: Old schedule:

AGV1 : T11, T23, T00
AGV2 : T21, T01, T10
New schedule:

AGV1: T23, T11, T00
AGV2: T21, T10, T01

5) Mutation 5: Randomly pick two normal tasks, one from each of the AGV's, with the longest finishing time and the shortest finishing time under the same schedule, and exchange them.

Ex: Old schedule:

AGV1: T23, T11, T00 (AFT1 = 45)
AGV2: T24, T22, T10 (AFT2 = 60)
AGV3: T10, T00, T25 (AFT3 = 39)

New schedule:

AGV1: T23, T11, T00
AGV2: T25, T22, T10
AGV3: T10, T00, T24

T25 and T24 are exchanged.

6) Mutation 6: Randomly choose a normal task from an AGV and replace it with a randomly selected new path.

Ex: Old AGV1: T22, T10, T01

New AGV1: T24, T10, T01
7) Mutation7: Since the AGV’s have different travelling speeds, we might improve the total finishing time of any schedule, by exchanging all the tasks between the AGV’s with the longest and shortest finishing time.

Ex: Old schedule:

AGV 1: T21,T00,T01 (AFT1=60)
AGV 2: T11,T24,T11 (AFT2=45)
AGV 3: T23,T11,T00 (AFT3=40)

New schedule:

AGV 1: T23,T11,T00
AGV 2: T11,T24,T11
AGV 3: T21,T00,T01

These 7 mutation operators are used to generate new schedules. However, the application of these operators are controlled by probabilities.

4.4 Reproduction

The reproduction process is typically based on the fitness values of the schedules. The reasoning is that schedules with higher fitness values should have higher probability of surviving to the next generation. In general, we can use a biased roulette wheel to execute the reproduction operator. The roulette wheel is divided
into slots, and each schedule in the population occupies a number of slots proportional to its fitness value. We then randomly generate numbers, as an index, into the wheel to determine which schedule will be retained to the next generation. Schedules with higher fitness values have larger space in the wheel, and are more likely to be chosen and retained to the next population. We will slightly modify this process by keeping the best schedule and throw away the worst one. The reproduction procedure is listed in the following:

1) Calculate the fitness value of each schedule in an old population (containing \(v\) schedules) and sum all their fitness values, \(S\). The fitness value is computed as \(1/T\) (\(T\) is the total finishing time of a schedule).

2) Retain the schedule with the best fitness value.

3) Each schedule occupies a number of slots in a roulette wheel proportional to its fitness value.

4) Repeat the following steps \(v\) times. Randomly select one number, between 0 and \(S\), and put the schedule occupying that slot into the new population.

5) After doing step 4, a new population is generated.

6) Throw away the schedule with the worst fitness value from the new population, and add the schedule with the best fitness value.

To understand more clearly the above procedure, we cite a small example below.

Given a population of 5 schedules; 2 AGV's

<table>
<thead>
<tr>
<th>AGV</th>
<th>Schedule</th>
<th>Fitness Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGV0: 40 (AFT0)</td>
<td>SFT0 = 40</td>
<td></td>
</tr>
<tr>
<td>AGV1: 35 (AFT1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AGV0: 50 (AFT0)
Schedule 1: SFT1 = 50
AGV1: 40 (AFT1)

AGV0: 37 (AFT0)
Schedule 2: SFT2 = 45
AGV1: 45 (AFT1)

AGV0: 56 (AFT0)
Schedule 3: SFT3 = 56
AGV1: 32 (AFT1)

AGV0: 32 (AFT0)
Schedule 4: SFT4 = 41
AGV1: 41 (AFT1)

F.V_j: Fitness value of schedule j
F.V0 = 1/40; F.V1 = 1/50; F.V2 = 1/45; F.V3 = 1/56; F.V4 = 1/40
F.V0+F.V1+F.V2+F.V3+F.V4=F.V (Total fitness value)
A: The best schedule.

According to the fitness values, we can form a roulette wheel, and randomly select numbers between 0 and F.V. If the number is a slot occupied by C, then we retain schedule2 into the new population. We repeat this process until the new population has 5 schedules. Suppose the new population is {A,B,D,E,B}, then we modify the population to be {A,B,D,A,B} by adding the best schedule, A, and throwing away
the worst schedule, E. We also can randomly select mutation operators to quickly process toward the optimal task schedule.

4.5 Complete algorithm

1. **Build up an initial population:**
   a) Randomly build an initial population by using coarse arrangement.
   b) Modify the population by using fine arrangement.

2. **Reproduction:**
   a) Compute the fitness value for every schedule and keep the schedule with the best fitness value.
   b) Construct a roulette wheel according to the fitness values of every schedule.
   c) Randomly select new schedules from the roulette wheel to form a new population.
   d) Add the best schedule to the new population and remove the worst one from the new population.

3. **Mutation operators:**
   a) Each mutation operator is associated with a stochastic assignment.
   b) Based on the stochastic assignment, we randomly apply one mutation operator.
4. Jump to step 2 until convergent.
Chapter 5

Simulation results

5.1 Task description

Table 5.1 - 5.3 and Fig. 5.1 - 5.3 show a typical FMS with three AGV’s, seven workstations and three different types of tasks: T0, T1 and T2. The number next to each workstation is the amount of execution time taken in the workstation.

<table>
<thead>
<tr>
<th></th>
<th>W0</th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
<th>W4</th>
<th>W5</th>
<th>W6</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>W1</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>W2</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>W3</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>W4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>0</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>W5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>W6</td>
<td>9</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.1

(The travelling time of AGV0 between workstations)

<table>
<thead>
<tr>
<th></th>
<th>W0</th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
<th>W4</th>
<th>W5</th>
<th>W6</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>W1</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>W2</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>W3</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>W4</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>W5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>W6</td>
<td>8</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.2

(The travelling time of AGV1 between workstations)
Table 5.3
(The travelling time of AGV2 between workstations)

<table>
<thead>
<tr>
<th>W0</th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
<th>W4</th>
<th>W5</th>
<th>W6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 5.1 T0 graph

Fig. 5.2 T1 graph

Fig. 5.1 T2 graph
We will modify T0, T1 and T2 from Fig. 5.1, Fig. 5.2 and Fig. 5.3 respectively with the help of data from Table 5.1 - 5.3 into the following:

e: executing time in an assigned workstation.

t: travelling time between two workstations.

wi: workstation i.

---->: the travelling direction of AGV's.

| T00: w0---->w2---->w4---->w0 | T01: w0---->w2---->w3---->w0 |
| e t e t e t e e t e t e t e e | e t e t e t e e t e t e t e t e |
| AGV0 4 5 6 7 7 6 2 | AGV0 4 5 6 4 8 8 2 |
| AGV1 4 5 6 6 7 6 2 | AGV1 4 5 6 4 8 7 2 |
| AGV2 4 5 6 6 7 5 2 | AGV2 4 5 6 4 8 6 2 |

| T10: w0---->w6---->w2---->w5---->w0 |
| e t e t e t e t e t e t e t e |
| AGV0 2 9 10 3 12 5 9 4 4 |
| AGV1 2 8 10 3 12 5 9 3 4 |
| AGV2 2 7 10 2 12 4 9 3 4 |

| T11: w0---->w2---->w6---->w5---->w0 |
| e t e t e t e t e t e t e t e |
| AGV0 2 5 12 3 10 8 9 4 4 |
| AGV1 2 5 12 3 10 7 9 3 4 |
| AGV2 2 5 12 2 10 7 9 3 4 |

| T20: w1---->w4---->w3---->w6---->w5---->w3---->w4---->w1 |
| e t e t e t e t e t e t e t e t e t e |
| AGV0 2 5 8 8 9 2 4 8 2 7 4 8 7 5 5 |
| AGV1 2 5 8 8 9 3 4 7 2 7 4 8 7 5 5 |
| AGV2 2 5 8 8 9 3 4 7 2 6 4 8 7 5 5 |
The above shows the execution time and travelling time for all possible paths and AGV’s. Based on the above 3 types of tasks and their paths, we will run our simulation with different number of tasks.
5.2 Tables for experiments

To study the performance of the genetic algorithm, we performed six different simulations for 2 AGV's.

a) \( T_0=10, T_1=10, T_2=6 \)
b) \( T_0=10, T_1=30, T_2=12 \)
c) \( T_0=10, T_1=10, T_2=20 \)
d) \( T_0=20, T_1=20, T_2=20 \)
e) \( T_0=30, T_1=30, T_2=30 \)
f) \( T_0=50, T_1=50, T_2=50 \)
Table 5.4

The last task arrangement:

\[
00, 01, 00, 11, 11, 11, 01, 01, 00, 24, 24, 24, 24 \ (AGV0) \\
24, 24, 11, 11, 01, 00, 11, 10, 10, 00, 10, 10, 00, 10 \ (AGV1)
\]

* _ : The task with double underline is a colliding task.

Note. For convenience, T is omitted from each task for the above schedule.
b): table:

$(T_0, T_1, T_2) = (10, 30, 12)$

# of schedules: 20

# of AGV's: 2

# of generations: 6000

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Finishing time</th>
<th>Iterations</th>
<th>Finishing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1537</td>
<td>4200</td>
<td>1466</td>
</tr>
<tr>
<td>200</td>
<td>1502</td>
<td>4400</td>
<td>1466</td>
</tr>
<tr>
<td>400</td>
<td>1493</td>
<td>4600</td>
<td>1466</td>
</tr>
<tr>
<td>600</td>
<td>1480</td>
<td>4800</td>
<td>1466</td>
</tr>
<tr>
<td>800</td>
<td>1478</td>
<td>5000</td>
<td>1466</td>
</tr>
<tr>
<td>1000</td>
<td>1478</td>
<td>5200</td>
<td>1466</td>
</tr>
<tr>
<td>1200</td>
<td>1478</td>
<td>5400</td>
<td>1466</td>
</tr>
<tr>
<td>1400</td>
<td>1472</td>
<td>5600</td>
<td>1466</td>
</tr>
<tr>
<td>1600</td>
<td>1472</td>
<td>5800</td>
<td>1466</td>
</tr>
<tr>
<td>1800</td>
<td>1469</td>
<td>6000</td>
<td>1466</td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td>1469</td>
<td></td>
</tr>
<tr>
<td>2200</td>
<td></td>
<td>1468</td>
<td></td>
</tr>
<tr>
<td>2400</td>
<td></td>
<td>1468</td>
<td></td>
</tr>
<tr>
<td>2600</td>
<td></td>
<td>1468</td>
<td></td>
</tr>
<tr>
<td>2800</td>
<td></td>
<td>1468</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td></td>
<td>1468</td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td></td>
<td>1468</td>
<td></td>
</tr>
<tr>
<td>3400</td>
<td></td>
<td>1468</td>
<td></td>
</tr>
<tr>
<td>3600</td>
<td></td>
<td>1467</td>
<td></td>
</tr>
<tr>
<td>3800</td>
<td></td>
<td>1466</td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td></td>
<td>1466</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.5

The last task arrangement:

11, 11, 11, 11, 11, 11, 24, 11, 11, 11, 24, 00, 24, 11, 00, 24, 24, 11, 01, 11, 01, 11, 00, 00 (AGV0)

24, 24, 24, 01, 11, 11, 11, 24, 24, 11, 24, 11, 24, 11, 10, 10, 11, 11, 11, 10, 10, 00, 10, 11, 01, 00, 11, 11 (AGV1)
c)

Table:

<table>
<thead>
<tr>
<th>(T0,T1,T2) = (10,10,20)</th>
<th># of schedules : 20</th>
<th># of AGV's : 2</th>
<th># of generations : 5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>Finishing time</td>
<td>Iterations</td>
<td>Finishing time</td>
</tr>
<tr>
<td>0</td>
<td>1275</td>
<td>4200</td>
<td>1207</td>
</tr>
<tr>
<td>200</td>
<td>1251</td>
<td>4400</td>
<td>1207</td>
</tr>
<tr>
<td>400</td>
<td>1237</td>
<td>4600</td>
<td>1207</td>
</tr>
<tr>
<td>600</td>
<td>1232</td>
<td>4800</td>
<td>1207</td>
</tr>
<tr>
<td>800</td>
<td>1229</td>
<td>5000</td>
<td>1207</td>
</tr>
<tr>
<td>1000</td>
<td>1228</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>1219</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1400</td>
<td>1219</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1600</td>
<td>1215</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1800</td>
<td>1215</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>1214</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2200</td>
<td>1213</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2400</td>
<td>1212</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2600</td>
<td>1211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2800</td>
<td>1211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>1211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3200</td>
<td>1208</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3400</td>
<td>1208</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3600</td>
<td>1207</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3800</td>
<td>1207</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>1207</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.6

The last task arrangement:

11, 24, 24, 11, 01, 11, 01, 11, 01, 24, 24, 01, 00, 01, 11, 24, 24, 24, 24, 24, 24, 01 (no collision)

24, 24, 22, 11, 01, 24, 24, 24, 24, 11, 01, 00, 24, 11, 24, 24, 10, 10, 20, 24 (no collision)
d) Table:

\[
(T_0, T_1, T_2) = (20, 20, 20)
\]

# of schedules : 20

# of AGV's : 2

# of generations : 10000

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Finishing time</th>
<th>Iterations</th>
<th>Finishing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1764</td>
<td>4200</td>
<td>1672</td>
</tr>
<tr>
<td>200</td>
<td>1712</td>
<td>4400</td>
<td>1672</td>
</tr>
<tr>
<td>400</td>
<td>1706</td>
<td>4600</td>
<td>1672</td>
</tr>
<tr>
<td>600</td>
<td>1702</td>
<td>4800</td>
<td>1672</td>
</tr>
<tr>
<td>800</td>
<td>1695</td>
<td>5000</td>
<td>1672</td>
</tr>
<tr>
<td>1000</td>
<td>1689</td>
<td>5200</td>
<td>1672</td>
</tr>
<tr>
<td>1200</td>
<td>1683</td>
<td>5400</td>
<td>1672</td>
</tr>
<tr>
<td>1400</td>
<td>1680</td>
<td>5600</td>
<td>1672</td>
</tr>
<tr>
<td>1600</td>
<td>1679</td>
<td>5800</td>
<td>1672</td>
</tr>
<tr>
<td>1800</td>
<td>1678</td>
<td>6000</td>
<td>1672</td>
</tr>
<tr>
<td>2000</td>
<td>1677</td>
<td>6200</td>
<td>1672</td>
</tr>
<tr>
<td>2200</td>
<td>1674</td>
<td>6400</td>
<td>1670</td>
</tr>
<tr>
<td>2400</td>
<td>1673</td>
<td>6600</td>
<td>1670</td>
</tr>
<tr>
<td>2600</td>
<td>1673</td>
<td>6800</td>
<td>1670</td>
</tr>
<tr>
<td>2800</td>
<td>1673</td>
<td>7000</td>
<td>1670</td>
</tr>
<tr>
<td>3000</td>
<td>1673</td>
<td>7200</td>
<td>1670</td>
</tr>
<tr>
<td>3200</td>
<td>1673</td>
<td>7400</td>
<td>1670</td>
</tr>
<tr>
<td>3400</td>
<td>1673</td>
<td>7600</td>
<td>1670</td>
</tr>
<tr>
<td>3600</td>
<td>1673</td>
<td>8000</td>
<td>1670</td>
</tr>
<tr>
<td>3800</td>
<td>1673</td>
<td>10000</td>
<td>1667</td>
</tr>
<tr>
<td>4000</td>
<td>1672</td>
<td>10000</td>
<td>1667</td>
</tr>
</tbody>
</table>

Table 5.7

The last task arrangement:

\[
11, 11, 24, 24, 01, 24, 00, 00, 11, 00, 11, 01, 00, 24, 11, 24, 11, 00, 11, 11, 24, 24, 00, 11, 00, 24, 24, 01, 10 \quad (\text{AGV}_0)
\]

\[
24, 11, 24, 24, 24, 00, 00, 00, 24, 24, 00, 11, 11, 01, 01, 00, 11, 24, 11, 00, 00, 24, 11, 10, 11, 00, 24, 24, 11, 23, 24 \quad (\text{AGV}_1)
\]
e) Table:

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Finishing time</th>
<th>Iterations</th>
<th>Finishing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2642</td>
<td>10500</td>
<td>2527</td>
</tr>
<tr>
<td>500</td>
<td>2601</td>
<td>11000</td>
<td>2527</td>
</tr>
<tr>
<td>1000</td>
<td>2583</td>
<td>11500</td>
<td>2527</td>
</tr>
<tr>
<td>1500</td>
<td>2575</td>
<td>12000</td>
<td>2525</td>
</tr>
<tr>
<td>2000</td>
<td>2571</td>
<td>12500</td>
<td>2525</td>
</tr>
<tr>
<td>2500</td>
<td>2568</td>
<td>13000</td>
<td>2525</td>
</tr>
<tr>
<td>3000</td>
<td>2560</td>
<td>13500</td>
<td>2521</td>
</tr>
<tr>
<td>3500</td>
<td>2559</td>
<td>14000</td>
<td>2512</td>
</tr>
<tr>
<td>4000</td>
<td>2541</td>
<td>14500</td>
<td>2509</td>
</tr>
<tr>
<td>4500</td>
<td>2536</td>
<td>15000</td>
<td>2509</td>
</tr>
<tr>
<td>5000</td>
<td>2534</td>
<td>15500</td>
<td>2509</td>
</tr>
<tr>
<td>5500</td>
<td>2534</td>
<td>16000</td>
<td>2509</td>
</tr>
<tr>
<td>6000</td>
<td>2534</td>
<td>16500</td>
<td>2509</td>
</tr>
<tr>
<td>6500</td>
<td>2531</td>
<td>17000</td>
<td>2508</td>
</tr>
<tr>
<td>7000</td>
<td>2528</td>
<td>17500</td>
<td>2508</td>
</tr>
<tr>
<td>7500</td>
<td>2528</td>
<td>18000</td>
<td>2508</td>
</tr>
<tr>
<td>8000</td>
<td>2528</td>
<td>18500</td>
<td>2505</td>
</tr>
<tr>
<td>8500</td>
<td>2528</td>
<td>19000</td>
<td>2505</td>
</tr>
<tr>
<td>9000</td>
<td>2527</td>
<td>19500</td>
<td>2505</td>
</tr>
<tr>
<td>9500</td>
<td>2527</td>
<td>20000</td>
<td>2505</td>
</tr>
<tr>
<td>10000</td>
<td>2527</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.8

The last task arrangement:

00, 01, 00, 01, 00, 00, 11, 24, 24, 11, 24, 24, 11, 00, 01, 00, 11, 11, 00, 24, 00, 24, 00, 11, 00, 24, 24, 11, 01, 11, 24, 24, 11, 24, 24, 11, 11, 00, 11, 11, 00, 11, 11, 00, 11, 11, 01 (AGV0)

24, 24, 24, 24, 00, 00, 00, 11, 11, 00, 24, 11, 24, 24, 24, 01, 01, 11, 00, 11, 11, 24, 24, 01, 11, 01, 11, 11, 01, 11, 10, 01, 24, 23, 11, 25, 24, 24, 24, 11, 10, 24, 24 (AGV1)
Table 5.9

The last task arrangement:
00, 24, 01, 00, 11, 01, 01, 01, 11, 11, 01, 11, 11, 11, 11, 00, 11, 00, 24, 11, 00, 00, 11, 24, 11, 01, 11, 11, 24, 24, 01, 24, 24, 10, 24, 24, 01, 24, 11, 01, 11, 00, 11, 01, 11, 01, 00, 24, 24, 24, 22, 00, 00, 24, 24, 00, 00, 00, 11, 00, 11, 01, 11, 11, 24, 24, 1, 24, 11, 24, 11, 00 (AGV0)
24, 24, 24, 10, 22, 24, 24, 24, 24, 23, 22, 24, 01, 01, 10, 24, 10, 00, 10, 00, 24, 24, 24, 01, 24, 24, 01, 11, 00, 24, 10, 24, 11, 01, 24, 11, 24, 10, 01, 01, 11, 11, 24, 01, 10, 11, 10, 00, 24, 24, 10, 24, 24, 01, 00, 24, 10, 24, 11, 24, 00, 11, 10, 11, 00, 24, 22, 01, 11, 11, 24, 22, 00, 11, 00, 10, 11, 00, 01, 00, 11, 24, 00, 11, 11, 01, 01, 24, 01, 11, 24, 24, 11, 23, 10, 10, 10, 10, 10, 21, 00, 23 (AGV1)
Next we perform the simulation with 3 AGV’s.

a) Table:

<table>
<thead>
<tr>
<th>(T0,T1,T2) = (10,10,6)</th>
<th># of schedules : 20</th>
<th># of AGV’s : 3</th>
<th># of generations : 1050</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>Finishing time</td>
<td>Iterations</td>
<td>Finishing time</td>
</tr>
<tr>
<td>0</td>
<td>490</td>
<td>540</td>
<td>459</td>
</tr>
<tr>
<td>30</td>
<td>481</td>
<td>570</td>
<td>459</td>
</tr>
<tr>
<td>60</td>
<td>480</td>
<td>600</td>
<td>459</td>
</tr>
<tr>
<td>90</td>
<td>477</td>
<td>630</td>
<td>459</td>
</tr>
<tr>
<td>120</td>
<td>467</td>
<td>660</td>
<td>459</td>
</tr>
<tr>
<td>150</td>
<td>462</td>
<td>690</td>
<td>459</td>
</tr>
<tr>
<td>180</td>
<td>460</td>
<td>720</td>
<td>458</td>
</tr>
<tr>
<td>210</td>
<td>459</td>
<td>750</td>
<td>458</td>
</tr>
<tr>
<td>240</td>
<td>459</td>
<td>780</td>
<td>458</td>
</tr>
<tr>
<td>270</td>
<td>459</td>
<td>810</td>
<td>458</td>
</tr>
<tr>
<td>300</td>
<td>459</td>
<td>840</td>
<td>458</td>
</tr>
<tr>
<td>330</td>
<td>459</td>
<td>870</td>
<td>458</td>
</tr>
<tr>
<td>360</td>
<td>459</td>
<td>900</td>
<td>458</td>
</tr>
<tr>
<td>390</td>
<td>459</td>
<td>930</td>
<td>458</td>
</tr>
<tr>
<td>420</td>
<td>459</td>
<td>960</td>
<td>458</td>
</tr>
<tr>
<td>450</td>
<td>459</td>
<td>990</td>
<td>458</td>
</tr>
<tr>
<td>480</td>
<td>459</td>
<td>1020</td>
<td>458</td>
</tr>
<tr>
<td>510</td>
<td>459</td>
<td>1050</td>
<td>458</td>
</tr>
</tbody>
</table>

Table 5.10

The last task arrangement:

10,11,11,01,11,11,10,23 (AGV0)
24,11,00,10,24,00,01,01,11 (AGV1)
00,23,25,01,11,24,00,00,01 (AGV2)
b) Table:

<table>
<thead>
<tr>
<th>(T0, T1, T2) = (10, 10, 20)</th>
<th># of schedules: 20</th>
<th># of AGV’s: 3</th>
<th># of generations: 3000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterations</td>
<td>Finishing time</td>
<td>Iterations</td>
<td>Finishing time</td>
</tr>
<tr>
<td>0</td>
<td>869</td>
<td>2160</td>
<td>815</td>
</tr>
<tr>
<td>120</td>
<td>842</td>
<td>2280</td>
<td>815</td>
</tr>
<tr>
<td>240</td>
<td>837</td>
<td>2400</td>
<td>815</td>
</tr>
<tr>
<td>360</td>
<td>834</td>
<td>2520</td>
<td>815</td>
</tr>
<tr>
<td>480</td>
<td>825</td>
<td>2640</td>
<td>815</td>
</tr>
<tr>
<td>600</td>
<td>825</td>
<td>2760</td>
<td>815</td>
</tr>
<tr>
<td>720</td>
<td>825</td>
<td>2880</td>
<td>815</td>
</tr>
<tr>
<td>840</td>
<td>825</td>
<td>3000</td>
<td>815</td>
</tr>
<tr>
<td>960</td>
<td>819</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1080</td>
<td>819</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>819</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1320</td>
<td>819</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1440</td>
<td>818</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1560</td>
<td>817</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1680</td>
<td>815</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1800</td>
<td>815</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1920</td>
<td>815</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2040</td>
<td>815</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.11

The last task arrangement:

24, 24, 24, 24, 00, 01, 23, 24, 25, 00, 11, 11, 01 (AGV0)
11, 10, 24, 24, 00, 11, 01, 00, 11, 24, 23, 24, 24, 01 (AGV1)
24, 24, 00, 00, 11, 24, 10, 22, 24, 10, 10, 24, 24 (AGV2)
The results in Tab. 5.4-5.11 are plotted in Fig. 5.4-5.11 respectively.
Fig. 5.6

\((T_0, T_1, T_2) = (10, 10, 20); 2 \text{ AGVs}\)

Fig. 5.7

\((T_0, T_1, T_2) = (20, 20, 20); 2 \text{ AGVs}\)
Fig. 5.8

$(T_0, T_1, T_2) = (30, 30, 30); 2 \text{ AGV's}$

Fig. 5.9

$(T_0, T_1, T_2) = (70, 70, 70); 2 \text{ AGV's}$
Fig. 5.10
(T0,T1,T2) = (10,10,6); 3 AGV's

Fig. 5.11
(T0,T1,T2) = (10,10,20); 3 AGV's
5.3 Discussions

From the simulation results, we observe that

1) The number of collisions in 2 AGV's is less than that in 3 AGV's from our simulation results. This is because 3 AGV's has a larger probability to enter the same workstation at the same time than 2 AGV's.

2) Since more collisions can occur in 3 AGV's, it is more difficult to reduce the total finishing time of schedules by our mutation operators.

Typically, genetic algorithms rely on a crossover operator to generate new search nodes (schedules) whereas mutation operators play an auxiliary role. In this thesis, the mutation operators play the most important roles and crossover operators are never used. The reasons are:

1) There are at least 2 AGV's in every schedule, thus performing mutation operations on the tasks in each AGV is similar to performing crossover operations.

2) When any two tasks from different schedules are exchanged, illegal schedules are very likely to be generated. This is because a task may be duplicated or missing after crossover operations. Therefore, we do not use crossover operations.
Task assignments are numerous when the number of tasks is large. Every task can be assigned to any position in any schedule. According to the simulation results, an optimal solution has the following properties:

1) Avoid task collisions, because collisions will increase AGV waiting time.
2) If there are "OR" or "AND" tasks, the paths with the shorter task finishing time must be used.
3) If the first workstation of a task and the last workstation of another is the same, then travelling time will be eliminated by scheduling them together.
4) If the subtasks needs to visit the same workstation more than once, it is better to schedule these visits to gather.
5) When assigning tasks to AGV's, it is better to assign them to fastest AGV.
Chapter 6

Conclusion

In this thesis, we studied the task scheduling problem for a flexible manufacturing system using genetic algorithms. The FMS consists of m workstations, p AGV's and n tasks. The tasks are further divided into AND, OR and SINGLE types. The task scheduling problem can then be stated as finding the optimal routing assignment of the p AGV's among the m workstations such that n tasks can be complete in the shortest time.
References


Appendix

Software Listing

Total Pages: 18
```c
#include<stdio.h>
#include<math.h>

#define part
#define tk kind
#define oav
#define agv
#define salia
#define max bh
#define mac_agv
#define max tk
#define non-bhtk
#define wk kind

/* parameter definition */
char bh_kind[16] = {2,2,6,0,0,0,0,0,0,0,0,0,0,0,0,0};
/*no of branch of each task*/
int ehtk n0[16] = {14,14,14,0,0,0,0,0,0,0,0,0,0,0,0,0};
/*no of each task*/

char agv_kind[max agv] = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1};
/*no of agv exist*/

#define bh cont
#define max bh
#define max tk
#define non-bhtk
#define wk kind

/* Abbreviation area(important)
workstation(wkst)
e(e.t)execute time / t(t.t)•travel time
If(t*finish time)
stringemtrasche/branch task/tk•task/conttwo content(wkst,e.t or t.t),
occolision
max agv
max. agv's in every schedule*/

char tk00[nam agv] = bh contte({0,4,5,2,6,7,4,0,2,0,2,0,2,2,2,2});
char tk01[nam agv] = bh contte({0,4,3,2,6,4,3,8,8,0,2,0,2,2,2,2});
char tk10[max agv] = bh contte({0,2,9,6,10,3,2,12,5,5,9,4,0,4,0});
char tk11[max agv] = bh contte({0,2,5,2,12,3,6,10,7,5,9,3,0,4,0});
char tk20[max agv] = bh contte({0,9,7,4,1,8,7,3,1,5,4,5,4,7,5,5});
char tk21[max agv] = bh contte({0,9,7,4,1,8,7,3,1,5,4,5,4,7,5,5});
char tk22[max agv] = bh contte({0,9,7,4,1,8,7,3,1,5,4,5,4,7,5,5});
char tk23[max agv] = bh contte({0,9,7,4,1,8,7,3,1,5,4,5,4,7,5,5});
char tk24[nam agv] = bh contte({0,9,7,4,1,8,7,3,1,5,4,5,4,7,5,5});
char tk30[max agv] = bh contte({0,4,2,6,7,4,7,4,2,0,2,0,2,2,2,2});

char travO[wk kind][wk kind] = {
    {10,1,3,8,6,4,9,1},
    {1,0,4,3,5,6,4,2},
    {5,4,0,4,6,5,3,2},
    {7,3,4,0,8,7,5,4},
    {6,5,6,5,0,6,1,3},
    {3,6,5,7,5,0,7,4},
    {4,4,3,3,6,7,0,1}};

char travl[wk kind][wk kind] = {
    {1,2,5,4,8,4,3,2,6,3,4,3,6,4,3,3,9,0,4,7,5,1,5,0}};
char trav2[wk kind][wk kind] = {
    {0,1,5,6,4,5,2,6,3,4,0,3,9,3,6,4,5,4,7,5,1,5,0}};
char trav3[wk kind][wk kind] = {
    {1,2,5,4,8,4,3,2,6,3,4,3,6,4,3,3,9,0,4,7,5,1,5,0}};
char trav4[wk kind][wk kind] = {
    {1,2,5,4,6,4,5,2,6,3,4,0,3,9,3,6,4,5,4,7,5,1,5,0}};
```

The content of every branch task in any schedule is restored in the array via different agv's/*
char col_bbh(schd no)(strg no)(max_tk);/*The array is used to restore the collided branch tasks of strings, in a schedule with shortest f.t*/char best_bbh[strg no][max_tk];/*The array is used to restore the arrangement branch tasks of any string in a schedule with shortest f.t*/

int best_schd;/*The schedule with shortest f.t*/char beta_col_bbh[strg no][max_tk];/*The array used to restore the collided branch tasks of strings, in a schedule with shortest f.t*/

int new_schd[achd no];/*New generated schedules are restored in the array*/int old_schd(schd no);/* 7 */

int kkk7away_fg,
unsigned int times_voy
int probabi[6][w(767,717,700,703,694,711),
int oper_no="nain()/*
int remain;
/*printf("ninit_string");*/
init_string();/*
*/
printf("nnsodify_init_string"),
*/
nodify_init_string(),
*/
printf("ncount_complet_tmO*");
*/
count Nopelele;*/
*/
printf("nreproduationO*),
*/
reproduction(),
*/
printf("nreproduceatO*)
*/
reproduceat(),
*/
printf("nmodify_schedule 
*/
modify_schedule()
*/
printf("%modify_reproduction\n");
/ *
*/
modify_reproduct();
/ *
*/
printf("%modify_schedule\n");
/ *
*/
modify_schedule();
/ *
*/
printf("%modify_schedule\n");
/ *
*/
search_bug();
/ *
*/
show_bug();
/ *
*/
display_best();
/ *
*/
test_know();
/ *
*/
int s,b,h,allth[kind];
for(wv) scanned_nodes();
for(avg) setkind(s++);
allth[kind]=wv;
for(wv) category_node();
for(bug) clean_threads();
{ bnode_order[a][a][b];
if(b>max_kinds)
{ switch(b)
  case 1;
  allth[1]=allth[0]+break;
  case 10;
  case 20;
  case 30;
  case 40;
  default:
  printf("%move entities,seed,seed,seed,seed,time_set",n_.a,b,h,allth,kind_no);exit(0);
  }
  }
  if(allth=0)
  goto good_apg;
  else
goto count_thugs;
  }
  count_thugs();
if(allth[0]==allth[0])
{ printf("%trouble in t2_node,seed,seed,seed,seed",allth[0],allth[1],allth[2]);exit(0);
if(allth[1]==allth[1])
{ printf("%trouble in t2_node,seed,seed,seed,seed",allth[0],allth[1],allth[2]);exit(0);
if(allth[2]==allth[2])
{ printf("%trouble in t2_node,seed,seed,seed,seed",allth[0],allth[1],allth[2]);exit(0);
}
}
display_best();
{ int s,b,h;
  printf("%in best nods");
  for(avg) cat_node(){
    printf("%n");
  for(bug) drain_MESH();
    printf("%n");
  }
  }
  printf("%in best nods");
  for(avg) cat_node(){
    printf("%n");
  for(bug) drain_MESH();
    printf("%n");
  }
  printf("%n");
  }
  search_best();
  { int s,b,h;
    for(avg) scanned_nodes();
    for(bug) drain_MESH();
    { bnode_order[a][a][b];
      if(b>max_kinds)
      { switch(b)
        case 1;
        allth[1]=allth[0]+break;
        case 10;
        case 20;
        case 30;
        case 40;
        default:
        printf("%move entities,seed,seed,seed,seed,time_set",n_.a,b,h,allth,kind_no);exit(0);
      }
    }
  }
  operator_finsh();
  { int 0;
    int b0,b0,kind_max[4];
    //
    printf("%in operator_pgn");
    for(avg) sc node() {
      kind_max[a]=0;
      body;
      for(avg) Action();
        printf("%move entities,seed,seed,seed,seed,time_set",n_.a,b,h,allth,kind_no);exit(0);
    }
  }
  / *
The content of every branch task in any schedule is restored in the array via different keys:

```c
char col[80] = [schedule][weight][branch task][collided branch tasks are restored in the array] /*(schedule)(weight)(branch task)*/
```

The arrangement of branch tasks in every string is restored in the array:

```c
int main_string(schedule no) [string] /*(schedule)(string)*/
```

The ft's of t00, t10 and t20 are restored in the array:

```c
int main_ft(int schedule no) [string no][2] /*The ft of every schedule is restored in the array*/
```

The ft of every branch task in any schedule is restored in the array:

```c
int branch_ft(schedule no) [branch task] /*The ft of every branch task in any schedule is restored in the array*/
```

The ft of every branch task in any schedule is restored in the array:

```c
int branch_ft(schedule no) [branch task] /*The ft of every branch task in any schedule is restored in the array*/
```

The arrangement of branch tasks in every string is restored in the array:

```c
int branch_arrangement(string no) /*(schedule)(branch task)*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft's of t00, t10 and t20 are restored in the array:

```c
int main_ft(schedule no) [string no][2] /*The ft of every string is restored in the array*/
```

The ft's of t00, t10 and t20 are restored in the array:

```c
int main_ft(schedule no) [string no][2] /*The ft's of t00, t10 and t20 are restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```

The ft of every string is restored in the array:

```c
int string_ft(schedule no) [string no] /*The ft of every string is restored in the array*/
```
/*
  modify_reproduction();
  
  search() ;
  
  / *
  research_bugs();
  
  search();
  */
  
  display_best();exit(0);
}

/*
  / *
  modify_schedule();
  
  search();
  */
  
  show_error();

/ */

int n,s,h,b,alltk[tk_kind];
for(s=task_name++)
  for(h=task_name++)
    for(b=task_name++)
      { alltk[0]=0;
        for(b=task_name++)
          { allt[alltk[0]]++;break;
            case 0: allt[alltk[0]]++;break;
            case 21: allt[alltk[0]]++;break;
            case 22: allt[alltk[0]]++;break;
            case 23: allt[alltk[0]]++;break;
            case 24: allt[alltk[0]]++;break;
            case 25: allt[alltk[0]]++;break;
            default:
              printf("("count, setd.setd, bed, timesetd",
                     h.s.a.b, timesetd)/exit(0);
          }
        else
          { if(countSadly)
            goto countSadly;
            else
              goto count Sadly;
          }
      }

iframe()
int ot,lot,book,seeip ama(d);
/*
  / *
  / */
  for(m=0pr<61.44)
    cool naximile0/
  
  b.Cor
  fono...01.<41,.+44/.adi
  the meoipsoaalo of all
  f.tfo
  of
  'very
  sohd in a soseardh
  med.*/
  roctip max[s]e10000/probabilsoll
printf("%d ");
printf("@r 0,

for(i.0;i.<46,m++)/*build a biased roulette wheel consisting of 100 equal parts*/
( recip max(s)•(recip_eax(s)*100/b),
/*every schd occupies some continue parts stored in recip mix array*/

printf("%d",recip_eax(s)];
if(b > k)
break;
if(open_note)
open_no++;
printf("%d",open_no);
)
debug(msg)
{ int a,b;
for(setd(recip_eax++)
for(setd(recip_eax++)
 if(b0_orders[a][b]=new_nbhk)
 { if(b0_orders[a][b]=new_nbhk)
 (show_outlk());
 printf("%s",a,k,b);return 0;
 }
}
update6()/*exchange normal nbhs in every settings*/
 { int t0=(open_no);
 t0=(open_no).
 stop_fg(0)
 printf("%s",open_no);
 printf("%s",open_no);
 printf("%s",open_no);
 for(setd(recip_eax++)
 { new_nbhs[b](b+1);
 for(setd(recip_eax++)
 { new_nbhs[b](b+1);
 0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
}
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(b0_orders[a][b]=new_nbhk)
{ if(b0_orders[a][b]=new_nbhk)
 t0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 }
if(times_no>4)
for(setd(recip_eax++)
 { times_no++;0=(open_no);
 stop_fg(0)
 printf("%s",open_no);
 })
printf("\n new col\n\n for(aw0,e<sodul no,s++)
 for(am0,a<strg no,a+*)
 {printf (\\n\")'!
 for(te0,bosax tk,b++)
 printf("02d
\w .col bh(sI(a)(1),)
 /*replaaft the worst sohd with the bust .chd*/
 modify_reproduct()
 { int e,teup,a,
 /*Prihtf("\nold new soh\n\n yor(s.0,e<schd_noTe++)
 printf(" Ild",new_woh(e1),
 printf("\t"),
 /*esp•oln..dhd ne.1,/vsehd "a" is the worst
 find minter
 1
,5r(sw0;e<sehd nolo++)
 ( if(new sehTa)aweehtm orderra1113)
 /*find -the worst •clea from all edhda./
 (teepol,breakt
 )
 if(templw1)
 --a1/.if
 no 'reset
 malld exiets,ge
 on finding
 worse ■ehd */
 if(alw0)
 goto find mints.,
 all
 -schdo are best,abie/
 goto not_ahange,
 )
 new eahrs)weabtal_erder(0)(1),
 /*preplaoe the worst eohd
 in all new
 generated solids with best sohd/!
 not change:,
 /*Printf (*Armee newpah")
 for(p.0.<.0hd_no).44.)
 printf("td",new_soh(e1)/*/
 /*max sahtm(e)(a)./
 /*arrIoge the order of schds aoacrding to the f.t of
every schd*/
 arrang_wohta_order()
 { int 1,,b,a,a7
 int eintaw0,
 for(sw0,e<eohd
 note++)
 for(aw0,a<2,...)
 { sehte_
 longest/*aax_sahtm store the
 f.t in every ■ohd*/
 aw0p
 for(se0,,,Csand noie++)
 eintew30000p
 forflow0,b<stbd
 orb++)
 ifchie_order(b)(0)<w
 sante)
 a/*find sehd with ehereest f.t*/
 mintewsalite_order(h) (0)
 )
 gotta order(m)(0).30000,oehtm_order(e)r1)...,
 /*sta.
 the solid and find next shorter sand./
 )
 for(ew0,e<sabd no,o++)
 achta_ordoral(0)wmax schtm((eabta_arder[N][1]))(0),
 /*•troe the f.t of Zil 'Wads*/
 printf("Noschte_order"),
 for(se0,0<aandno,s++)
 yor(ow0,a<2,e++)
 /\n printf(" td",schtm_erderlra)),
 /*ea•_echtm[N](a)•/
 /.nase the good schds(geod
 genes)
 to next research nodes*/
 reproduction()
 { int s, d,
 mitotic float ...sip masjeohd
 for(swOlataandno,s++)/*idd the realproaals of all
 f.t's of every scald in
 •vumarah node•/
 ( recip_max(e).1.0/0.0Leohtm(s)(0),
 bwh+recip ■axis),
 printf(" 45.31,be05.5f
* .r.aip wortml,b),
)
 for(e.0,e<seb4 note++)/*bulid • biased roulette wheel
 eansieting of
 100 small parts*/
 w.(w).(recip max(0/1)*100,
 /*every redid eeoupies some ~tiny., parts stored
 in reaip_mox
 array*/
 printf("15.2f"),x.nir
 max(.)),
 /*am6S536/(2.100.0),*/
 for(e.0,e<sehd_ne,d++)
 ( beloirea4Pjaax(d)+0.0,
 if (b > k)
 (noW_sahte)md,eld eahlalwdtbrwoat,
 /*put
 the new generated sated into new 'lab
-ar*/
 priority(*Nr
)
```c
// do
for(a=0; a<strg_no; a++)
    { if (stop_fp(a) == 1)  
        goto :.find;  
    if(d >= strg_no)  
        /*if the strg_no. of collided strgs is 1 or 0, skip*/
        continue;
    maxi=0;  
    for(aw0; a<strg_no+a++)  
        { if(d > strg_no-1)  
            goto :.find;  
        if((str_tm2(s)(a)>max) ti  
            { stop_fg(t)=0;  
                continue;  
            }  
        maxotx_ta2[a]=0;  
        for(aw0; a<strg_no+a++)  
            { if((str_tm2(s)(a)>max) ti  
                continue;  
            }  
        maxo=
```
comp(a)ret,
comp feed
else=0
for(i=0;i<entry_num;i++)
for(s=0;s<entry_num;s++)
if((entry_fp[i][s]<>0) & (entry_ta[i][s]<min)
return(entry_fp[i][s])
/*Count the f.t. if all steps before last step are counted*/
else
return(-1)
/*Count the f.t. of the last step of every entry*/

if((entry_fp[i][s]<>0) & (entry_ta[i][s]<min)
if((entry_fp[i][s]<>0) & (entry_ta[i][s]<min)

if(((stop fp[i][s]<>0) & (entry_ta[i][s]<min))
else
if(((stop fp[i][s]<>0) & (entry_ta[i][s]<min))
/*count the f.t. of all steps after last step*/

for(i=0;i<entry_num;i++)
if(rtart_fp[i][s]<min)
/*The f.t. of 1st hit in any strg is counted*/

if(min••30000)
/*if f.t. of any strg in a sched is overcome, do next sched*/

if(ags_fp[i][s]<>0)
/*the previous count is to add*/

for(i=0;i<entry_num;i++)
if(alwa2)
if((oomP(a).* 1 )/*find one strg in longest f.t from collided strgs*/
if(st. t.1.1[41 > maartrtm)
maxstrtmmn►
for(amOsa<strg no/a++)
if(alwa2)
if(oomP(a).* 1 )/*find one strg in longest f.t from collided strgs*/

/*mod; bhth being counting.*/
if(mamatrtm > str_tm(s)(a21)
/*collision is happoned, so avv must wait*/

if(tk(h)1a21(02+1)1■0)/*take 1.0*/

if(tk(h)(42)(02+3)1.0)
/*next what exists*/

else/*Ith(1.1(.21(02+31mw0)/nemt wk.t doesn't exist*/

else/*comp filmO►ne collision/directly add executing to*/

/*store col blotk*/

else/*comp fpil 0*/
/*if(tk(h)2lenon bbtk)
/*next what doesn't exist*/

else/*stop counting this strg*/
```c
/*
 for(r•0/x<kinds r++)
 for(a•0/0;kinds c++)
 tmp[r][a]=tmp2[r][0];
 */

for(r•0/x<kinds r++)
 for(a•0/0<kinds c++)
 trav[tx][a][c]=trav2[tx][a][c];
 */

printf("%d",tx[0][0]+1);
printf("\n");
for(co0/30/p++)
 printf("%d",travfo[co][0])
 printf("\n");
for(coOpoptwk kind/c++)
 printf("%d",travfo[co][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
for(o.Ops<salad no/a++)
 printf("%d",travfo[o][0]);
printf("\n");
```
// Add a task to task_order
if (tk_order[s][a] == 0)
    tk_order[s][a] = tkno;
else
    randomize();

print("%d, %d, %d, %d, \n", tk_order[s][a], tkno[2], tkno[1], tkno[0]);
```c
/*

*/

int i, j;
int a, a, m, b, temp, b2, k;
int ft[.trag no], min tm, d, m, ft0, ftl, tm0, tml, tm00, tmlit;

for (a = 0;a < trg no ++)
for (b = 0;b < max thy/a ++)
printf("%d", a);

for (i = 0;i < 2;i++)
for (j = 0;j < max th[tj;++)
printf("%d", i);

/*

*/

else/• bh ardem[al](b2)j
if (abs((ft[a0]-tm0+tm11) -
( ftrall-tml+tm00)) < "ab((ft[a11...ft[a0])))
if (tm0 > tali)
if (abs((ft[a0]-tm0+tm11) -
( ftrall-tml+tm00)) <

*/

else/• bh ardem[al](b2)j
if (abs((ft[a0]-tm0+tm11) -
( ftrall-tml+tm00)) <

*/

```