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

APPROVAL SHEET

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

List of Figures

Listing of Tables

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 performs 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 are 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.).

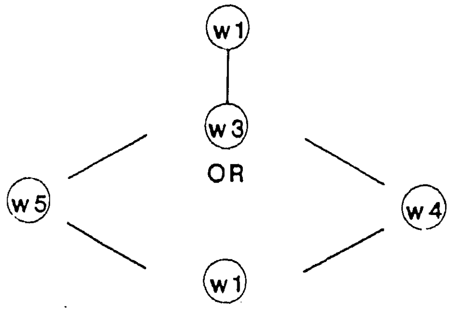


Fig. 2.1
OR task
(T0)

T0 has two possible paths: T00 and T01

T00: w1-->w3-->w5-->w1

T01: w1-->w3-->w4-->w1

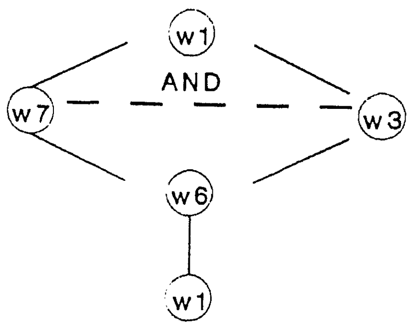


Fig. 2.2
AND task
(T1)

T1 has two possible paths: T10 and T11

T10: w1-->w7-->w3-->w6-->w1

T11: w1-->w3-->w7-->w6-->w1

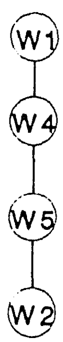


Fig. 2.3
SINGLE task
(T2)

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={ A1,A2, ... , Ap }, where Ai indicates the ith AGV; W={ W1, W2, ... , Wm }, where Wi indicates workstation i; T={ T1, T2, ... , Tk }, where Ti indicates the ith type of task and N={ N1, N2, ... , Nk }, where Ni indicates the number of the ith 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 :

$$ET\,j = TET\,1j + TET\,2j + \dots + TET\,kj$$

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:

$$TT\,j = TT\,1j + TT\,2j + \dots + TT\,kj$$

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:

$$WT\,j = WT\,1j + WT\,2j + \dots + WT\,kj$$

From the above definitions, the total finishing time of T1 through Tk 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 (T1,T2, ... , Tk). Then the total finishing time of the ith schedule (i is between 1 and v) is defined as:

$$SFT\,i = \max \{ TA1, TA2, \ldots , TAp \}$$

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

$$BSFT = \min \{ SFT1, SFT2, \ldots , SFTv \}$$

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

```
T0     w1--->w4--->w6--->w2
       e  t  e  t  e  t  e
AGV0   3  4  6  3  7  6  3

T1     w0--->w3--->w5--->w2
       e  t  e  t  e  t  e
AGV1   2  3  7  4  9  3  2
```

Let us calculate the total finishing time of the schedule by using the following and data :

```
Starting point of AGV's             Ending point of AGV0
     |                                          |
     0   3     7    13   16    23     29  30    33

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

$$ET0 = 3 + 6 + 7 + 3 = 19 \,;\, ET1 = 2 + 7 + 9 + 2 = 20$$

$$TT0 = 4 + 3 + 6 = 13 \,;\, TT1 = 3 + 4 + 3 = 10$$

$$WT0 = 1 \,;\, WT1 = 0$$

$$TA0 = ET0+TT0+WT0=33 \;;\; TA1=ET1+TT1+WT1=30$$

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.

e) Generation : An iteration in genetic algorithm.

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 (T1 + T2 + ... + Tn), 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 = TFT1 + TFT2 + ... + TFTn. 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 = TFT1 + TFT2 + TFT3 + TFT4 + TFT5

=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 inital task arrangements, it can be modified to produce better task arrangements. For example,

AGV 1 : 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 | 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.

10) j=j+1; If j = n+1, then go to step11. Otherwise go to the above step 7.

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

12) u=u+1; If u=v, then stop. Otherwise 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 :

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

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

nl=2, ns=3

Step 1. AGV L=AGV1; AGV S=AGV2; LAFT=AFT1 and SAFT=AFT2.

Step2. T1 is the ith (i=1) task in AGV1; T3 is the jth (j=1)task in AGV2.

Step3. Exchange the two tasks in step2, so the new task arrangement becomes the following:

AGV1: T3-->T2 (NLAFT=12+20=32)

AGV2: T1-->T4-->T5 (NSAFT=30+10+8 = 48)

LAFT - SAFT = 50 - 30 = 20; NSAFT - NLAFT = 48 - 32 = 16

Step4. Since | LAFT - SAFT | > | NLAFT - NSAFT |, we accept the new arrangement and LAFT=48 and SAFT=32.

Step5. i=i+1; Since i (=2) is not equal to m+1 (= 3), go to step2 below.

Step2. T2 is the ith (i=2) task in the AGV1; T1 is the jth (j=1)task in the AGV2.

Step3. Exchange the two tasks in step2, so the new task arrangement becomes the following:

AGV1: T3-->T1 (NLAFT=12+30=42)

AGV2: T2-->T4-->T5 (NSAFT=20+10+8 = 38)

LAFT-SAFT=48-32=16; NSAFT - NLAFT = 42 - 38 = 4

Step4. Since | LAFT - SAFT | > | NLAFT - NSAFT |, LAFT=42 and SAFT=38.

Step5. i=i+1; Since i (=3) is equal to m+1 (= 3), go to step6.

Step6. i=1, j=1, AGV L = AGV1, AGV S = AGV2, LAFT=42, SAFT=38 and the task ordering is as follows:

AGV1: T3-->T1 (NLAFT=12+30=42)

AGV2: T2-->T4-->T5 (NSAFT=20+10+8 = 38)

Step7. T3 is the ith (i=1) task in the AGV1; T2 is the jth (j=1) task in the AGV2.

Step8. Exchange the two tasks in step7, so the new task arrangement becomes the following:

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

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

| LAFT-SAFT | = 42 - 38 = 4; | NSAFT - NLAFT | = 50 - 30 = 20

Step9. Since | LAFT - SAFT | < | NLAFT - NSAFT |, LAFT=42 and 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 mutation1. 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) Mutation6: 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)

AGV3 : 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 occuping 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

Schedule0 : AGV0 : 40 (AFT0)
AGV1: 35 (AFT1)
SFT0 = 40

Schedule1 : AGV0 : 50 (AFT0) / AGV1 : 40 (AFT1) SFT1 = 50

Schedule2 : AGV0 : 37 (AFT0) / AGV1 : 45 (AFT1) SFT2 = 45

Schedule3 : AGV0 : 56 (AFT0) / AGV1 : 32 (AFT1) SFT3 = 56

Schedule4 : AGV0 : 32 (AFT0) / AGV1 : 41 (AFT1) SFT4=41

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)

F.V0/F.V=A; F.V1/F.V=B; F.V2/F.V=C; F.V3/F.V=D; F.V4/F.V=E

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 step2 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.

	W0	W1	W2	W3	W4	W5	W6
W0	0	1	5	8	6	4	9
W1	1	0	4	3	5	6	4
W2	5	4	0	4	7	5	3
W3	8	3	4	0	8	7	2
W4	6	5	7	8	0	6	8
W5	4	6	5	7	6	0	8
W6	9	4	3	2	7	8	0

Table 5.1

(The travelling time of AGV0 between workstations)

	W0	W1	W2	W3	W4	W5	W6
W0	0	1	5	7	6	3	8
W1	1	0	4	3	5	6	4
W2	5	4	0	4	6	5	3
W3	7	3	4	0	8	7	3
W4	6	5	6	8	0	5	6
W5	3	6	5	7	5	0	7
W6	8	4	3	3	6	7	0

Table 5.2

(The travelling time of AGV1 between workstations)

	W0	W1	W2	W3	W4	W5	W6
W0	0	1	5	6	5	3	7
W1	1	0	4	3	5	6	4
W2	5	4	0	4	6	4	2
W3	6	3	4	0	8	6	3
W4	5	5	6	8	0	4	5
W5	3	6	4	6	4	0	7
W6	7	4	2	3	5	7	0

Table 5.3
(The travelling time of AGV2 between workstations)

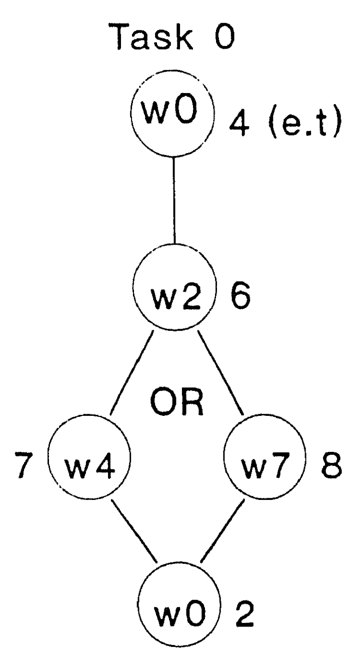


Fig. 5.1 T0 graph

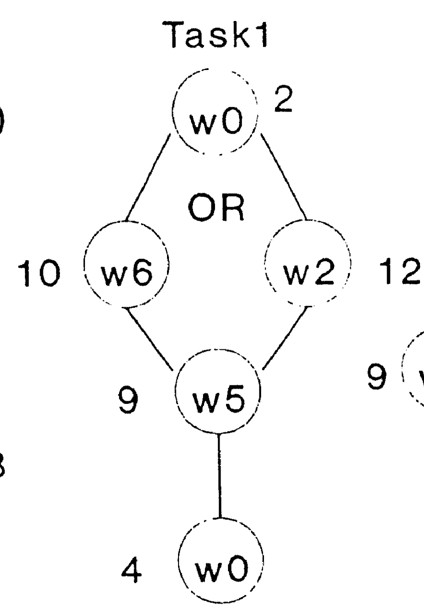


Fig 5.2 T1 graph

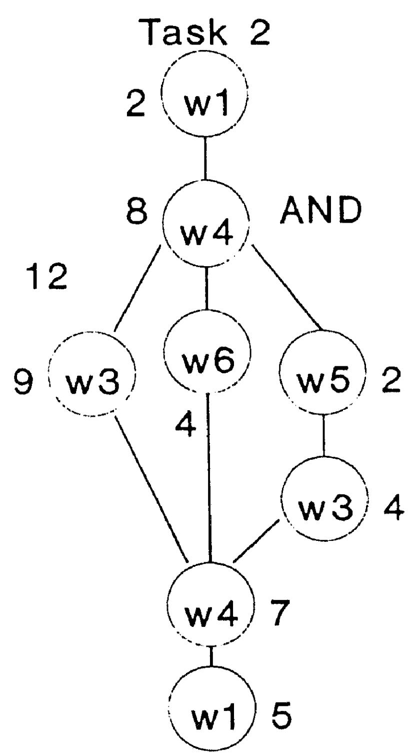


Fig. 5.1 T2 graph

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

	e	t	e	t	e	t	e
AGV0	4	5	6	7	7	6	2
AGV1	4	5	6	6	7	6	2
AGV2	4	5	6	6	7	5	2

T01: w0--->w2--->w3--->w0

	e	t	e	t	e	t	e
AGV0	4	5	6	4	8	8	2
AGV1	4	5	6	4	8	7	2
AGV2	4	5	6	4	8	6	2

T10: w0--->w6--->w2--->w5--->w0

	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
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
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

```
T21: w1--->w4--->w6--->w3--->w5--->w3--->w4--->w1
     e  t  e  t  e  t  e  t  e  t  e  t  e  t  e
AGV0 2  5  8  7  4  2  9  7  2  7  4  8  7  5  5
AGV1 2  5  8  6  4  3  9  7  2  7  4  8  7  5  5
AGV2 2  5  8  5  4  3  9  6  2  6  4  8  7  5  5

T22: w1--->w4--->w6--->w5--->w3--->w3--->w4--->w1
     e  t  e  t  e  t  e  t  e  t  e  t  e  t  e
AGV0 2  5  8  7  4  8  2  7  4  0  9  8  7  5  5
AGV1 2  5  8  6  4  7  2  7  4  0  9  8  7  5  5
AGV2 2  5  8  5  4  7  2  6  4  0  9  8  7  5  5

T23: w1--->w4--->w5--->w3--->w6--->w3--->w4--->w1
     e  t  e  t  e  t  e  t  e  t  e  t  e  t  e
AGV0 2  5  8  6  2  7  4  2  4  2  9  8  7  5  5
AGV1 2  5  8  5  2  7  4  3  4  3  9  8  7  5  5
AGV2 2  5  8  4  2  6  4  3  4  3  9  8  7  5  5

T24: w1--->w4--->w5--->w3--->w3--->w6--->w4--->w1
     e  t  e  t  e  t  e  t  e  t  e  t  e  t  e
AGV0 2  5  8  6  2  7  4  0  9  2  4  7  7  5  5
AGV1 2  5  8  5  2  7  4  0  9  3  4  6  7  5  5
AGV2 2  5  8  4  2  6  4  0  9  3  4  5  7  5  5

T25: w1--->w4--->w3--->w5--->w3--->w6--->w4--->w1
     e  t  e  t  e  t  e  t  e  t  e  t  e  t  e
AGV0 2  5  8  8  9  7  2  7  4  2  4  7  7  5  5
AGV1 2  5  8  8  9  7  2  7  4  3  4  6  7  5  5
AGV2 2  5  8  8  9  6  2  6  4  3  4  5  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) T0=10,T1=10,T2=6

b) T0=10,T1=30,T2=12

c) T0=10,T1=10,T2=20

d) T0=20,T1=20,T2=20

e) T0=30,T1=30,T2=30

f) T0=50,T1=50,T2=50

a) :
Table :

(T0,T1,T2)=(10,10,6) # of schedules : 20 # of AGV's : 2 # of generations : 500			
Iterations	Finishing time	Iterations	Finishing time
0	712	380	686
20	710	400	686
40	707	420	686
60	703	440	686
80	700	460	686
100	699	480	686
120	696	500	686
140	696		
160	696		
180	696		
200	693		
220	693		
240	693		
260	693		
280	690		
300	690		
320	689		
340	687		
360	686		

Table 5.4

The last task arrangement :

```
00,01,00,11,11,11,01,01,00,24,24,24,24 ( AGV0 )
24,24,11,11,01,00,11,10,10,00,10,00,10 ( 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 :

(T0,T1,T2)=(10,30,12) # of schedules : 20 # of AGV's : 2 # of generations : 6000			
Iterations	Finishing time	Iterations	Finishing time
0	1537	4200	1466
200	1502	4400	1466
400	1493	4600	1466
600	1480	4800	1466
800	1478	5000	1466
1000	1478	5200	1466
1200	1478	5400	1466
1400	1472	5600	1466
1600	1472	5800	1466
1800	1469	6000	1466
2000	1469		
2200	1468		
2400	1468		
2600	1468		
2800	1468		
3000	1468		
3200	1468		
3400	1468		
3600	1467		
3800	1466		
4000	1466		

Table 5.5

The last task arrangement :

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

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

c)
Table :

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

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,
24,24,01 ( no collision )

24,24,22,11,01,24,24,24,11,01,00,24,11,24,24,10,10,20,
24 ( no collision )
```

d)
Table :

(T0,T1,T2)=(20,20,20) # of schedules : 20 # of AGV's : 2 # of generations : 10000			
Iterations	Finishing time	Iterations	Finishing time
0	1764	4200	1672
200	1712	4400	1672
400	1706	4600	1672
600	1702	4800	1672
800	1695	5000	1672
1000	1689	5200	1672
1200	1683	5400	1672
1400	1680	5600	1672
1600	1679	5800	1672
1800	1678	6000	1672
2000	1677	6200	1672
2200	1674	6400	1670
2400	1673	6600	1670
2600	1673	6800	1670
2800	1673	7000	1670
3000	1673	7200	1670
3200	1673	7400	1670
3400	1673	7600	1670
3600	1673	7800	1670
3800	1673	8000	1670
4000	1672	10000	1667

Table 5.7

The last task arrangement :

```
11,11,24,24,01,24,00,00,11,00,11,11,01,00,24,11,24,11,
00,11,11,24,24,00,11,00,24,24,01,10 ( AGV0 )

24,11,24,24,24,00,00,00,24,00,11,11,01,01,00,11,24,11,
00,00,24,11,10,11,00,24,24,11,23,24 ( AGV1 )
```

e)
Table :

(T0,T1,T2)=(30,30,30) # of schedules : 20 # of AGV's : 2 # of generations : 20000			
Iterations	Finishing time	Iterations	Finishing time
0	2642	10500	2527
500	2601	11000	2527
1000	2583	11500	2527
1500	2575	12000	2525
2000	2571	12500	2525
2500	2568	13000	2525
3000	2560	13500	2521
3500	2559	14000	2512
4000	2541	14500	2509
4500	2536	15000	2509
5000	2534	15500	2509
5500	2534	16000	2509
6000	2534	16500	2509
6500	2531	17000	2508
7000	2528	17500	2508
7500	2528	18000	2508
8000	2528	18500	2505
8500	2528	19000	2505
9000	2527	19500	2505
9500	2527	20000	2505
10000	2527		

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,
00,24,00,11,00,24,24,11,01,11,24,24,24,11,24,11,11,00,11,11,
00,00,11,00,11,11,01 ( 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 )
```

f) Table :

(T0,T1,T2)=(70,70,70) # of schedules : 20; # of AGV's : 2 # of generations : 16000			
Iterations	Finishing time	Iterations	Finishing time
0	6171	8400	5893
400	6043	8800	5890
800	6005	9200	5889
1200	5997	9600	5889
1600	5979	10000	5889
2000	5968	10400	5888
2400	5964	10800	5881
2800	5958	11200	5880
3200	5951	11600	5880
3600	5937	12000	5879
4000	5932	12400	5873
4400	5931	12800	5870
4800	5923	13200	5870
5200	5919	13600	5869
5600	5915	14000	5867
6000	5908	14400	5867
6400	5907	14800	5867
6800	5906	15200	5867
7200	5905	15600	5867
7600	5899	16000	5867
8000	5896		

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,
00,00,11,24,11,01,11,24,24,01,24,24,10,24,24,01,24,00,11,11,
00,01,23,24,11,11,01,00,24,11,01,11,01,00,01,00,24,00,24,24,
00,10,10,24,11,11,01,11,11,24,00,00,11,01,00,24,23,24,11,11,
01,00,24,24,24,22,00,00,24,24,00,00,00,11,00,11,11,01,11,11,
24,24,1,24,11,24,11,00 (AGV0)

24,24,24,10,22,24,24,24,23,22,24,01,01,10,24,10,00,10,00,24,
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,
11,10,11,00,00,24,22,01,11,11,24,22,00,11,00,10,11,00,01,00,
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 :

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

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 :

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

Table 5.11

The last task arrangement :

24,24,24,23,00,01,23,24,25,00,11,11,01(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. 5.4

(T0,T1,T2) = (10,10,6); 2 AGV's

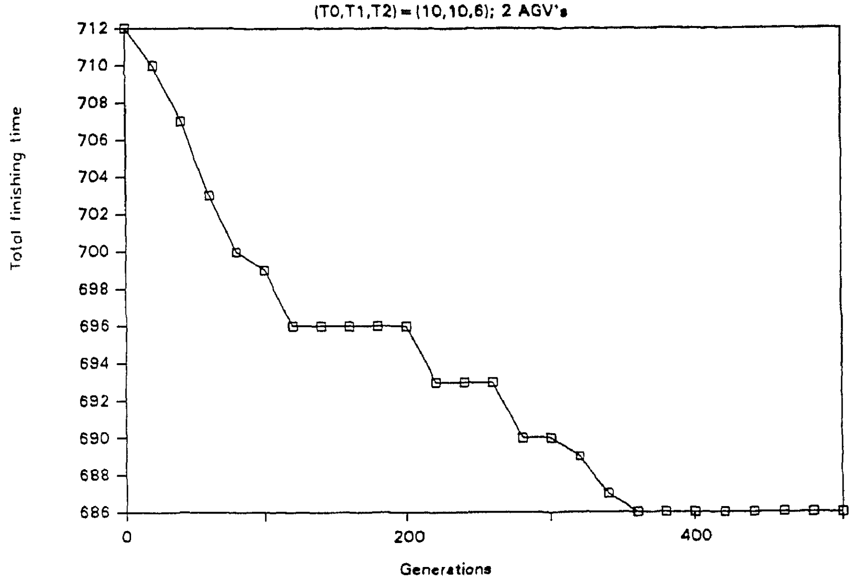


Fig. 5.5

(T0,T1,T2) = (10,30,12); 2 AGV's

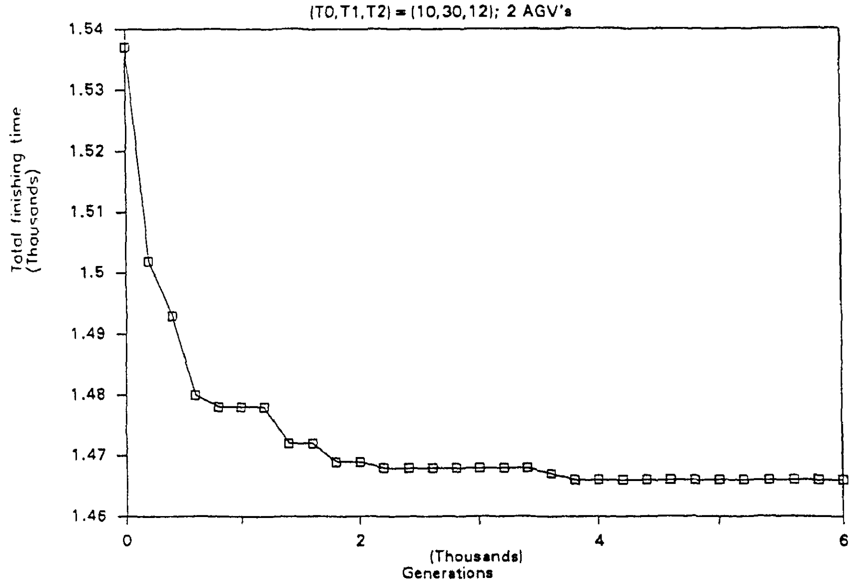


Fig. 5.6

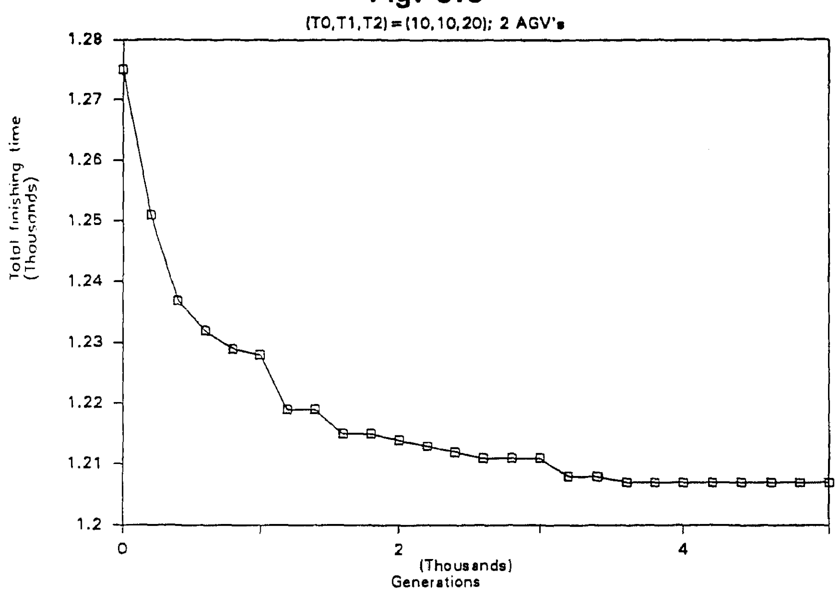

(T0,T1,T2) = (10,10,20); 2 AGV's
Total finishing time
(Thousands)
1.28
1.27
1.26
1.25
1.24
1.23
1.22
1.21
1.2
0
2
4
(Thousands)
Generations

Fig. 5.7

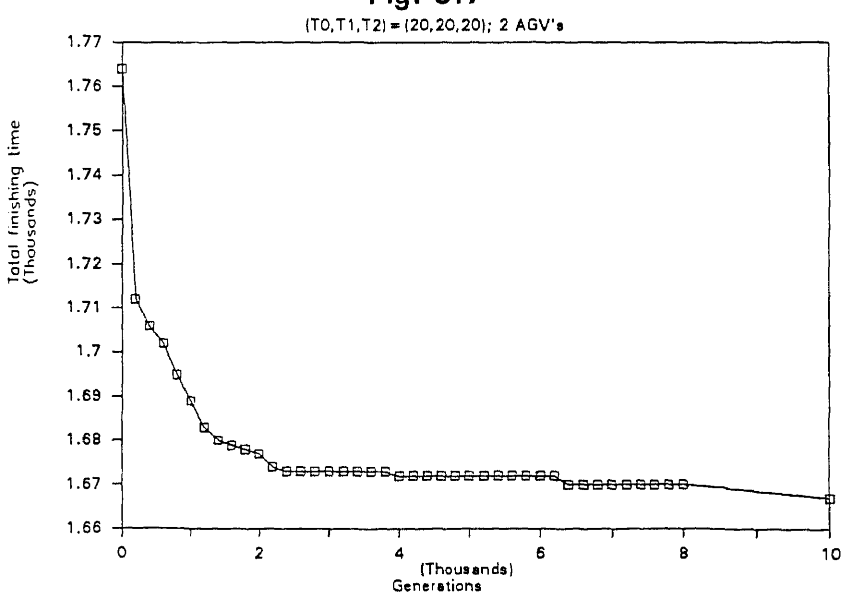

(T0,T1,T2) = (20,20,20); 2 AGV's
Total finishing time
(Thousands)
1.77
1.76
1.75
1.74
1.73
1.72
1.71
1.7
1.69
1.68
1.67
1.66
0
2
4
6
8
10
(Thousands)
Generations

Fig. 5.8

(T0,T1,T2) = (30,30,30); 2 AGV's

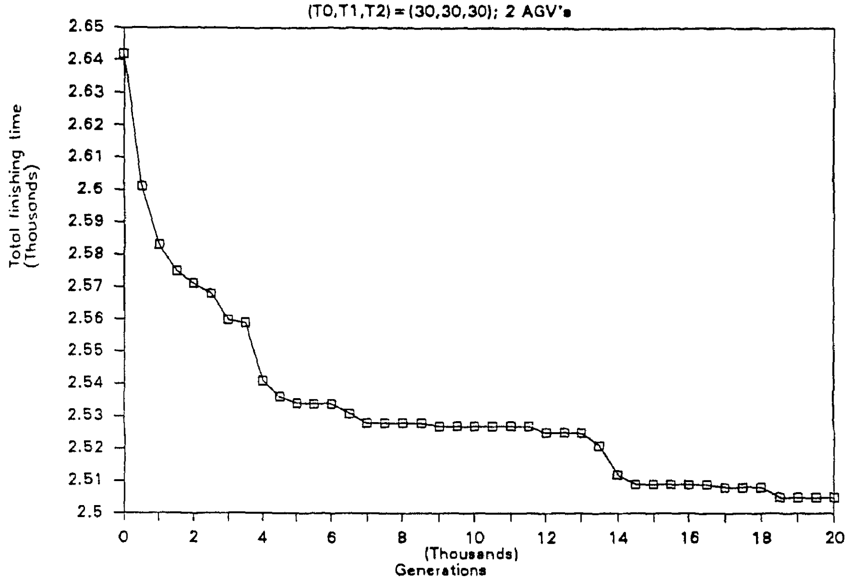


Fig. 5.9

(T0,T1,T2) = (70,70,70); 2 AGV's

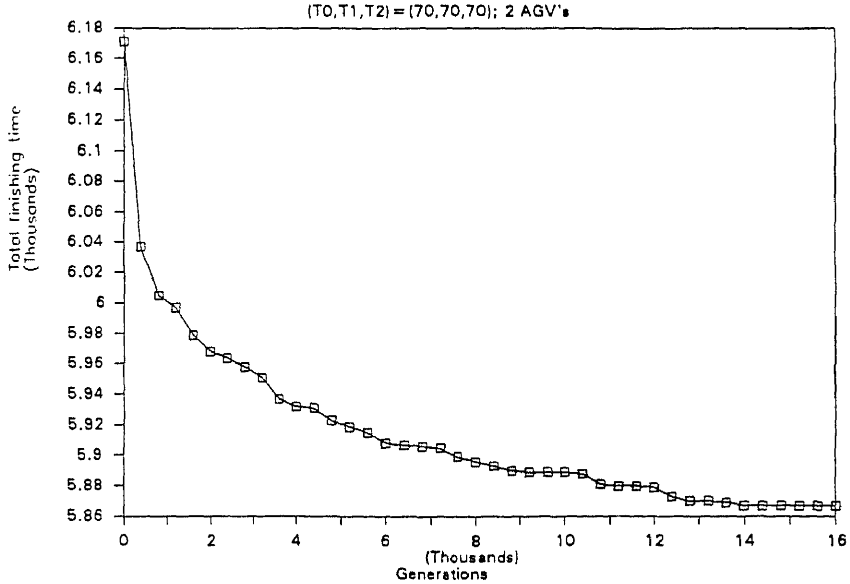


Fig. 5.10

(T0,T1,T2)=(10,10,6); 3 AGV's

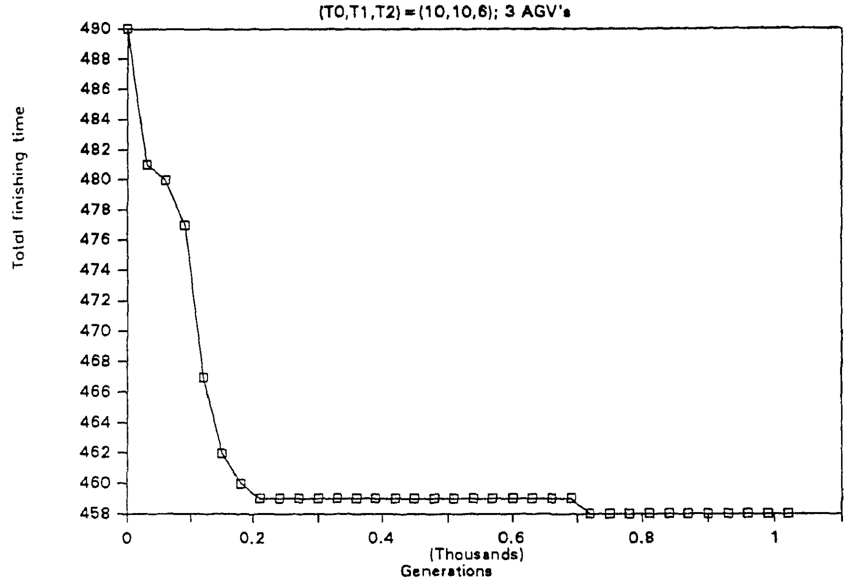
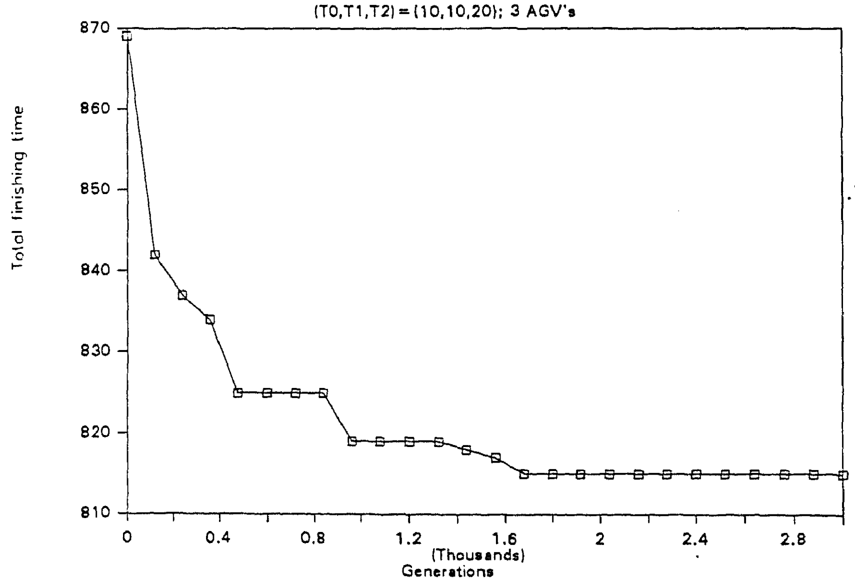


Fig. 5.11

(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 sechedules 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

[1] J. A. Simpson, R. J. Hocken, and J. S. Albus, "The automated manufacturing research facility of the National Bureau of Standards," *J. Manuf., Syst.*, vol. 1, no. 1, pp17-32, 1982.

[2] P. E. Chen and J. Talavage, "Production decision support system for computerized manufacturing systems," *J. Manuf. Syst.*, vol. 1, no. 2, pp.157-168, 1982.

[3] R. Sui and C. K. Whitney, "Decision support requirements in flexible manufacturing," *J. Manuf. Syst.*, vol. 3, no. 1, pp. 61-69, 1984.

[4] A. Ballakur and H. J. Steudel, "Integration of job shop control systems: A state-of-art review," *J. Manuf. Syst.*, vol. 3, no. 1, pp. 71-88, 1984.

[5] Y. C. Ho and X. R. Cao, "Performance sensitivity to routing changes in queuing networks and flexible manufacturing systems using perturbation analysis," *IEEE J. Robotics Automation*, vol. RA-1, no. 4, pp. 165-172 Dec. 1985.

[6] K. E. Stecke and T. L. Morin, "The optimality of balancing workload in certain types of flexible manufacturing systems," *Europ. J. Oper. Res.*, vol. 20, pp. 68-72, 1985.

[7] C. K. Whitney, "Integration in FMS design and control," *Proc. ASME Computers in Engineering Conf.*, 1986, pp. 355-360.

[8] R. E. Young, "Planning and control requirements for flexible manufacturing systems," in *Proc. ASME C omputers in Engineering Conf.*, 1986, pp. 347-353.

[9] O. Berman and O. Maimon, "Cooperation among flexible manufacturing systems," *IEEE J. Robotics Automation*, vol RA-2, NO. 2, PP. 24-30, Mar. 1986.

[10] C. L. Chen, C. S. George Lee and C. D. McGILLEM, "Task assignment and load balancing of autonomous vehicles in a flexible manufacturing system," *IEEE J. of Robotics and Automation*, vol. RA-3, no. 6, PP. 659-671, Dec. 1987.

[11] P. S. Lui and L. C. Fu, "Planning and scheduling in a flexible manufacturing system using a dynamic routing method for automated guided vehicles," *Intl. Conf. on Robotics & Automation*, pp. 1584-1589, 1989 .

[12] M. P. Fourmain, "Compaction of symbolic layout using genetic algorithms," *Proc. Intl. Conf. on Genetic Algorithms and their applications*, pp. 141-153, July 1985.

[13] J. I. Grefenstette, R. Gopal, B. J. Rosmaita, and D. Van Gucht, "Genetic algorithms for the traveling salesman problem," *Proc. Intl. Conf. Genetic algorithms and their Applications*, pp. 160-168, July 1985.

[14] L. B. Booker, "*Intelligent behavior as an adaptation to the task environment,*" Ph.D. thesis, Dept. Computer and Communication Sciences, Univ. of Michigan, Feb. 1982.

[15] S. F. Smith, "Flexible learning of problem solving heuristics through adaptive search," *Proc. of 8th IJCAI* 1983.

Appendix

Software Listing

Total Pages : 18

```
#include<stdio.h>
#include<math.h>
/**/
/* define part */
#define tk_kind 3/* 3 kinds of tasks(t0,t1,t2)*/
#define strg_no 3/* 2 strings in any schdule*/
#define agv_no 3/* 2 agv's in any schdule*/
#define schd_no 5/* 5 schdules in every reserch node*/
#define bh_contt 30/*30 bytes to store workstations,
excuting time and travelling time in any branch task*/
#define max_bh 6/* 7 */
#define max_agv 5/* max. agv's in every schdule*/
#define max_tk 50/* max. branch tasks in every string*/
#define non_bhtk 99/*99 means that t99 doesn't exist*/
#define max_bhtk 30/*t29(29);t29 is the max. bhanch task
from t00 to t29*/
#define wk_kind 7/* 7 kinds of workstations*/
/**/
 /* parameter definition */
char bh_kind[6]={2,2,6,0,0,0};
 /*no of branchs of each task*/
int ehtk_no[6]={40,40,40,0,0,0};
/*60,60,36[no of each task]*/
char agv_kind[max_agv]={1,1,1};
/* [agv no];1 means that agv exists*/
 /*abbreviation area(important)
 w=workstation(wkst) ; e(e.t)=execute time ; t(t.t)=travel
 time ;f.t=finishing time;string=strg;schedule=schd;
 bhtk=branch task;tk=task;contt=content(wkst,e.t or t.t);
 col=collision;normal bhtks=no collided bhtks;
 */
 /*agv0 and 1 do task0*/
 /*                               w e t w e t w e t w e t */
char tk00[max_agv][bh_contt]={{0,4,5,2,6,7,4,7,6,0,2,0},
                              {0,4,5,2,6,6,4,7,6,0,2,0},
                              {0,4,5,2,6,6,4,7,5,0,2,0}
                  };
 /*[agv no][content of task](The content means that
  workstation,executing time and travelling time*/
char tk01[max_agv][bh_contt]={{0,4,5,2,6,4,3,8,8,0,2,0},
                              {0,4,5,2,6,4,3,8,7,0,2,0},
                              {0,4,5,2,6,4,3,8,6,0,2,0}
                             };
/*agv0 and 1 do task1*/
/*          w e t w e  t w e  t w e t w e t*/
char tk10[max_agv][bh_contt]=
        {{0,2,9,6,10,3,2,12,5,5,9,4,0,4,0},
         {0,2,8,6,10,3,2,12,5,5,9,3,0,4,0},
         {0,2,7,6,10,2,2,12,4,5,9,3,0,4,0}
        };
char tk11[max_agv][bh_contt]=
        {{0,2,5,2,12,3,6,10,8,5,9,4,0,4,0},
         {0,2,5,2,12,3,6,10,7,5,9,3,0,4,0},
         {0,2,5,2,12,2,6,10,7,5,9,3,0,4,0}
        };
/*agv0 and 1 do task2*/
/*          w e t w e t w e t w e t w e t w e t w e t*/
char tk20[max_agv][bh_contt]=
        {{1,2,5,4,8,8,3,9,2,6,4,8,5,2,7,3,4,8,4,7,5,1,5,0},
         {1,2,5,4,8,8,3,9,3,6,4,7,5,2,7,3,4,8,4,7,5,1,5,0},
         {1,2,5,4,8,8,3,9,3,6,4,7,5,2,6,3,4,8,4,7,5,1,5,0}
        };
/*[agv no][workstation,executing time or travelling]*/
char tk21[max_agv][bh_contt]=
        {{1,2,5,4,8,7,6,4,2,3,9,7,5,2,7,3,4,8,4,7,5,1,5,0},
         {1,2,5,4,8,6,6,4,3,3,9,7,5,2,7,3,4,8,4,7,5,1,5,0},
         {1,2,5,4,8,5,6,4,3,3,9,6,5,2,6,3,4,8,4,7,5,1,5,0}
        };
char tk22[max_agv][bh_contt]=
        {{1,2,5,4,8,7,6,4,8,5,2,7,3,4,0,3,9,8,4,7,5,1,5,0},
         {1,2,5,4,8,6,6,4,7,5,2,7,3,4,0,3,9,8,4,7,5,1,5,0},
         {1,2,5,4,8,5,6,4,7,5,2,6,3,4,0,3,9,8,4,7,5,1,5,0}
        };
char tk23[max_agv][bh_contt]=
        {{1,2,5,4,8,6,5,2,7,3,4,2,6,4,2,3,9,8,4,7,5,1,5,0},
         {1,2,5,4,8,5,5,2,7,3,4,3,6,4,3,3,9,8,4,7,5,1,5,0},
         {1,2,5,4,8,4,5,2,6,3,4,3,6,4,3,3,9,8,4,7,5,1,5,0}
        };
char tk24[max_agv][bh_contt]=
        {{1,2,5,4,8,6,5,2,7,3,4,0,3,9,2,6,4,7,4,7,5,1,5,0},
         {1,2,5,4,8,5,5,2,7,3,4,0,3,9,3,6,4,6,4,7,5,1,5,0},
         {1,2,5,4,8,4,5,2,6,3,4,0,3,9,3,6,4,5,4,7,5,1,5,0}
        };
char tk25[max_agv][bh_contt]=
        {{1,2,5,4,8,8,3,9,7,5,2,7,3,4,2,6,4,7,4,7,5,1,5,0},
         {1,2,5,4,8,8,3,9,7,5,2,7,3,4,3,6,4,6,4,7,5,1,5,0},
         {1,2,5,4,8,8,3,9,6,5,2,6,3,4,3,6,4,5,4,7,5,1,5,0}
        };
char trav0[wk_kind][wk_kind]={ {0,1,5,8,6,4,9},
                               {1,0,4,3,5,6,4},
                               {5,4,0,4,7,5,3},
                               {8,3,4,0,8,7,2},
                               {6,5,7,8,0,6,7},
                               {4,6,5,7,6,0,8},
                               {9,4,3,2,7,8,0}
                 };/*[workstation a][workstation b];trav0
                      array means that
                      the time is taken from Wa to Wb under
                      agv0*/
char trav[strg_no][wk_kind][wk_kind];
char trav1[wk_kind][wk_kind]={ {0,1,5,7,6,3,8},
                               {1,0,4,3,5,6,4},
                               {5,4,0,4,6,5,3},
                               {7,3,4,0,8,7,3},
                               {6,5,6,8,0,5,6},
                               {3,6,5,7,5,0,7},
                               {8,4,3,3,6,7,0}
                 };/*[workstation a][workstation b];trav1
                      array means that the time is taken
                      from Wa to Wb by travelling via agv1*/
char trav2[wk_kind][wk_kind]={ {0,1,5,6,5,3,7},
                               {1,0,4,3,5,6,4},
                               {5,4,0,4,6,4,2},
                               {6,3,4,0,8,6,3},
                               {5,5,6,8,0,4,5},
                               {3,6,4,6,4,0,7},
                               {7,4,2,3,5,7,0}
                              };
char trav3[wk_kind][wk_kind];
char trav4[wk_kind][wk_kind];
/* */
char bhftsum[max_agv][30]={ {37,37,0,0,0,0,0,0,0,0,
                             58,57,0,0,0,0,0,0,0,0,
                             84,82,81,76,73,82},
                            {36,36,0,0,0,0,0,0,0,0,
                             56,55,0,0,0,0,0,0,0,0,
                             84,82,79,77,72,82},
                            {35,35,0,0,0,0,0,0,0,0,
                             53,54,0,0,0,0,0,0,0,0,
                             83,79,77,75,69,80}
                         };/*[agv_no][branch task];the f.t*/
char tk[66][max_agv][bh_contt];/*[branch task][agv][branch
```

```
                                        task content];The content
                                        of every branch task in
                                        any schdule is restored in
                                        the array via different
                                        agv's*/
char col_bh[schd_no][strg_no][max_tk];
 /*[schedule][string][branch task];collided branch tasks
   are restored in the array*/
int meantm;/*the rough f.t of every string*/
int mintm_err;
int tkbhsum[5];/*tk00sum,tk10sum,tk20sum,tk30sum,tk40sum*/
int tkftsum[3]={37,57,80};/*The f.t's of t00,t10 and t20
 are restored in the array*/
char tk_order[schd_no][320];/*[schedule][task no]*/
char bh_order[schd_no][strg_no][max_tk];
/*[schedule][string][branch];The arrangement of branch
 tasks in every string is restored in the array*/
int str_tm[schd_no][strg_no];
 /*[schedule][string];The f.t of every string is restored
   in the array*/
int str_tm2[schd_no][strg_no];
int ttkno=0;/*total no of tasks in every schdule*/
int sch_tm[schd_no][strg_no][2];
 /*The f.t of every schdule is restored in the array*/
int sch_tm2[schd_no][strg_no];
 /*The f.t of every schdule is restored in the array*/
int max_schtm[schd_no][strg_no];
 /*The max f.t of strings in any schdule is restored in
   the array*/
int schtm_order[schd_no][2];
 /*[schd no][tm value and tm priority];The array is used
   to restore the f.t of every schdule and schdule no.*/
int best_schd;/*The schdule with shortest f.t*/
char best_col_bh[strg_no][max_tk];
 /*The array Is used to estore the collided branch tasks
   of strings in a schdule with shortest f.t*/
char best_bh[strg_no][max_tk];
 /*The array is used to restore the arrangement branch
   tasks of any string in a schdule with shortest f.t*/
int best_strtm[strg_no];
int new_sch[schd_no];
 /*After reproduction,the new generated schdules are res-
   tored in the array*/
int old_sch[schd_no];/* ? */
int kkk,away_fg;
unsigned int times_no;
int probabi[6]={707,717,700,703,694,711};
int oper_no;
main()
{  int remain;
/*
   printf("\ninit_string");
*/
   init_string();
/*
   printf("\nmodify_init_string");
*/
   modify_init_string();
/*
   printf("\ncount_complet_tm0");
*/
   count_complet_tm();
/*
   printf("\narrang_schtm_order0");
*/
   arrang_schtm_order();
```

```
/*
   printf("\nreproduction0");
*/
   reproduction();
/*
   printf("modify_reproduct0");
*/
   modify_reproduct();
/*
   printf("\nmodify_schedule");
*/
   modify_schedule();
   for(times_no=0;times_no<60001;times_no++)
   {
     if(times_no==0)
       { printf("\n%d ",times_no);
         printf("%d,%d",best_schd,schtm_order[0][0]);
       }
     remain=times_no%1000;
     if((times_no>=1000)&&(remain==0))
       { printf("\n%d ",times_no);
         printf("%d,%d",best_schd,schtm_order[0][0]);
       }
     operater_probabi();
     switch(oper_no)
     {   case 0:mutate1();/*printf("\nmut1");*/break;
         case 1:mutate2();/*printf("\nmut2");*/break;
         case 2:mutate3();/*printf("\nmut3");*/break;
         case 3:mutate4();/*printf("\nmut4");*/break;
         case 4:mutate5();/*printf("\nmut5");*/break;
         case 5:mutate6();/*printf("\nmut6");*/break;
         default:break;
     }
/*
     printf("\ntest_tkno");
     test_tkno();
     printf("\nsearch_bug1");
     search_bug();
*/
/*
     printf("\ncount_complet_tm");
*/
     count_complet_tm();
/*
     printf("\nsearch_bug2");
     search_bug();
     printf("\ncatch_bug");
     catch_bug();
*/
/*
     printf("\narrang_schtm_order");
*/
     arrang_schtm_order();
/*
     printf("\nsearch_bug3");
     search_bug();
*/
/*
     printf("\nreproduction");
*/
     reproduction();
/*
     printf("\nsearch_bug4");
     search_bug();
*/
/*
```

```
        printf("\nmodify_reproduction");
*/
        modify_reproduct();
/*
        printf("\nsearch_bug5");
        search_bug();
*/
/*
        printf("\nmodify_schedule");
*/
        modify_schedule();
/*
        printf("\nsearch_bug6");
*/
        search_bug();
    }
/*
      show_colbh();
*/
      display_best();exit(0);
}
test_tkno()
{
   int s,a,b,h,alltk[tk_kind];
   for(s=0;s<schd_no;s++)
     {for(a=0;a<tk_kind;a++)
        alltk[a]=0;
      for(a=0;a<strg_no;a++)
       {for(b=0;b<max_tk;b++)
         { h=bh_order[s][a][b];
           if(h!=non_bhtk)
             { switch(h)
                { case 0:
                  case 1:alltk[0]=alltk[0]+1;break;
                  case 10:
                  case 11:alltk[1]=alltk[1]+1;break;
                  case 20:
                  case 21:
                  case 22:
                  case 23:
                  case 24:
                  case 25:alltk[2]=alltk[2]+1;break;
                  default:
                   printf("\nh=%d,s=%d,a=%d,b=%d,times=%d",
                          h,s,a,b,times_no);exit(0);
                }
             }
           else
             { if(a!=strg_no-1)
                  goto goon_agv;
                else
                 goto count_tkequ;
             }
         }
count_tkequ:
        if(alltk[0]!=ehtk_no[0])
         { printf("\ntrouble in t0_no=%d,s=%d,a=%d,b=%d\n",
                    alltk[0],s,a,b);
           printf("\nt1_no=%d,t2_no=%d",alltk[1],alltk[2]);exit(0);
         }
        if(alltk[1]!=ehtk_no[1])
         { printf("\ntrouble in t1_no=%d,s=%d,a=%d,b=%d\n",
                    alltk[1],s,a,b);
           printf("\nt0_no=%d,t2_no=%d",alltk[0],alltk[2]);exit(0);
         }
        if(alltk[2]!=ehtk_no[2])
```

```
         { printf("\ntrouble in t2_no=%d,s=%d,a=%d,b=%d\n",
                    alltk[2],s,a,b);
           printf("\nt0_no=%d,t1_no=%d",alltk[0],alltk[1]);exit(0);
         }
goon_agv:;
       }
     }
}
display_best()
{  int a,b;
   printf("\n best bh\n");
   for(a=0;a<strg_no;a++)
     { printf("\n");
       for(b=0;b<max_tk;b++)
         printf("%2d",best_bh[a][b]);
     }
   printf("\n");
   printf("\n best colbh");
   for(a=0;a<strg_no;a++)
     { printf("\n");
   .   for(b=0;b<max_tk;b++)
         printf("%2d",best_col_bh[a][b]);
     }
   printf("\n");
}
search_bug()
{
   int s,a,b,h;
   for(s=0;s<schd_no;s++)
     for(a=0;a<strg_no;a++)
       for(b=0;b<max_tk;b++)
         { h=bh_order[s][a][b];
           if(h!=non_bhtk)
            { switch(h)
               { case 0:
                 case 1:
                 case 10:
                 case 11:
                 case 20:
                 case 21:
                 case 22:
                 case 23:
                 case 24:
                 case 25:break;
                 default:
                   printf("\nh=%d,s=%d,a=%d,b=%d,times=%d",
                          h,s,a,b,times_no);
                   exit(0);
               }
            }
         }
}
operater_probabi()
{  int s;
   int b=0,c,k,recip_max[6];
/*
   printf("\n operater_pro");
*/
   for(s=0;s<6;s++)
      recip_max[s]=0;
   b=0;
   for(s=0;s<6;s++)/*add the reciprocals of all f.t's of
                     every schd in a research node*/
      { recip_max[s]=10000/probabi[s];
        b=b+recip_max[s];
/*
```

```
                                        task content];The content
                                        of every branch task in
                                        any schdule is restored in
                                        the array via different
                                        agv's*/
char col_bh[schd_no][strg_no][max_tk];
 /*[schedule][string][branch task];collided branch tasks
   are restored in the array*/
int meantm;/*the rough f.t of every string*/
int mintm_err;
int tkbhsum[5];/*tk00sum,tk10sum,tk20sum,tk30sum,tk40sum*/
int tkftsum[3]={37,57,80};/*The f.t's of t00,t10 and t20
 are restored in the array*/
char tk_order[schd_no][320];/*[schedule][task no]*/
char bh_order[schd_no][strg_no][max_tk];
/*[schedule][string][branch];The arrangement of branch
 tasks in every string is restored in the array*/
int str_tm[schd_no][strg_no];
 /*[schedule][string];The f.t of every string is restored
   in the array*/
int str_tm2[schd_no][strg_no];
int ttkno=0;/*total no of tasks in every schdule*/
int sch_tm[schd_no][strg_no][2];
 /*The f.t of every schdule is restored in the array*/
int sch_tm2[schd_no][strg_no];
 /*The f.t of every schdule is restored in the array*/
int max_schtm[schd_no][strg_no];
 /*The max f.t of strings in any schdule is restored in
   the array*/
int schtm_order[schd_no][2];
 /*[schd no][tm value and tm priority];The array is used
   to restore the f.t of every schdule and schdule no.*/
int best_schd;/*The schdule with shortest f.t*/
char best_col_bh[strg_no][max_tk];
 /*The array is used to estore the collided branch tasks
   of strings in a schdule with shortest f.t*/
char best_bh[strg_no][max_tk];
 /*The array is used to restore the arrangement branch
   tasks of any string in a schdule with shortest f.t*/
int best_strtm[strg_no];
int new_sch[schd_no];
 /*After reproduction,the new generated schdules are res-
   tored in the array*/
int old_sch[schd_no];/* ? */
int kkk,away_fg;
unsigned int times_no;
int probabi[6]={707,717,700,703,694,711};
int oper_no;
main()
{  int remain;
/*
   printf("\ninit_string");
*/
   init_string();
/*
   printf("\nmodify_init_string");
*/
   modify_init_string();
/*
   printf("\ncount_complet_tm0");
*/
   count_complet_tm();
/*
   printf("\narrang_schtm_order0");
*/
   arrang_schtm_order();

/*
   printf("\nreproduction0");
*/
   reproduction();
/*
   printf("modify_reproduct0");
*/
   modify_reproduct();
/*
   printf("\nmodify_schedule");
*/
   modify_schedule();
   for(times_no=0;times_no<60001;times_no++)
   {
     if(times_no==0)
       { printf("\n%d ",times_no);
         printf("%d,%d",best_schd,schtm_order[0][0]);
       }
     remain=times_no%1000;
     if((times_no>=1000)&&(remain==0))
       { printf("\n%d ",times_no);
         printf("%d,%d",best_schd,schtm_order[0][0]);
       }
     operater_probabi();
     switch(oper_no)
     {   case 0:mutate1();/*printf("\nmut1");*/break;
         case 1:mutate2();/*printf("\nmut2");*/break;
         case 2:mutate3();/*printf("\nmut3");*/break;
         case 3:mutate4();/*printf("\nmut4");*/break;
         case 4:mutate5();/*printf("\nmut5");*/break;
         case 5:mutate6();/*printf("\nmut6");*/break;
         default:break;
     }
 /*
     printf("\ntest_tkno");
     test_tkno();
     printf("\nsearch_bug1");
     search_bug();
 */
 /*
     printf("\ncount_complet_tm");
 */
     count_complet_tm();
 /*
     printf("\nsearch_bug2");
     search_bug();
     printf("\ncatch_bug");
     catch_bug();
 */
 /*
     printf("\narrang_schtm_order");
 */
     arrang_schtm_order();
 /*
     printf("\nsearch_bug3");
     search_bug();
 */
 /*
     printf("\nreproduction");
 */
     reproduction();
 /*
     printf("\nsearch_bug4");
     search_bug();
 */
 /*
```

```
      printf("\nmodify_reproduction");
*/
      modify_reproduct();
/*
      printf("\nsearch_bug5");
      search_bug();
*/
/*
      printf("\nmodify_schedule");
*/
      modify_schedule();
/*
      printf("\nsearch_bug6");
*/
      search_bug();
    }
/*
     show_colbh();
*/
     display_best();exit(0);
}
test_tkno()
{
   int s,a,b,h,alltk[tk_kind];
   for(s=0;s<schd_no;s++)
     {for(a=0;a<tk_kind;a++)
        alltk[a]=0;
      for(a=0;a<strg_no;a++)
       {for(b=0;b<max_tk;b++)
         { h=bh_order[s][a][b];
           if(h!=non_bhtk)
             { switch(h)
               { case 0:
                 case 1:alltk[0]=alltk[0]+1;break;
                 case 10:
                 case 11:alltk[1]=alltk[1]+1;break;
                 case 20:
                 case 21:
                 case 22:
                 case 23:
                 case 24:
                 case 25:alltk[2]=alltk[2]+1;break;
                 default:
                  printf("\nh=%d,s=%d,a=%d,b=%d,times=%d",
                         h,s,a,b,times_no);exit(0);
               }
             }
           else
             { if(a!=strg_no-1)
                 goto goon_agv;
               else
                goto count_tkequ;
             }
         }
count_tkequ:
        if(alltk[0]!=ebtk_no[0])
         { printf("\ntrouble in t0_no=%d,s=%d,a=%d,b=%d\n",
                   alltk[0],s,a,b);
           printf("\nt1_no=%d,t2_no=%d",alltk[1],alltk[2]);exit(0);
         }
        if(alltk[1]!=ebtk_no[1])
         { printf("\ntrouble in t1_no=%d,s=%d,a=%d,b=%d\n",
                   alltk[1],s,a,b);
           printf("\nt0_no=%d,t2_no=%d",alltk[0],alltk[2]);exit(0);
         }
        if(alltk[2]!=ebtk_no[2])
```

```
         { printf("\ntrouble in t2_no=%d,s=%d,a=%d,b=%d\n",
                   alltk[2],s,a,b);
           printf("\nt0_no=%d,t1_no=%d",alltk[0],alltk[1]);exit(0);
         }
goon_agv:;
       }
     }
}
display_best()
{  int a,b;
   printf("\n best bh\n");
   for(a=0;a<strg_no;a++)
     { printf("\n");
       for(b=0;b<max_tk;b++)
         printf("%2d",best_bh[a][b]);
     }
   printf("\n");
   printf("\n best colbh");
   for(a=0;a<strg_no;a++)
     { printf("\n");
       for(b=0;b<max_tk;b++)
         printf("%2d",best_col_bh[a][b]);
     }
   printf("\n");
}
search_bug()
{
   int s,a,b,h;
   for(s=0;s<schd_no;s++)
     for(a=0;a<strg_no;a++)
       for(b=0;b<max_tk;b++)
         { h=bh_order[s][a][b];
           if(h!=non_bhtk)
            { switch(h)
              { case 0:
                case 1:
                case 10:
                case 11:
                case 20:
                case 21:
                case 22:
                case 23:
                case 24:
                case 25:break;
                default:
                  printf("\nh=%d,s=%d,a=%d,b=%d,times=%d",
                         h,s,a,b,times_no);
                  exit(0);
              }
            }
         }
}
operater_probabi()
{  int s;
   int b=0,c,k,recip_max[6];
/*
   printf("\n operater_pro");
*/
   for(s=0;s<6;s++)
      recip_max[s]=0;
   b=0;
   for(s=0;s<6;s++)/*add the reciprocals of all f.t's of
                     every schd in a research node*/
     { recip_max[s]=10000/probabi[s];
       b=b+recip_max[s];
/*
```

```
        printf(" p0 ");
        printf(" rec=%d,b=%d,prob=%d  ",recip_max[s],b,
                probabi[s]);
*/
      }
   for(s=0;s<6;s++)/*bulid a biased roulette wheel
                     consisting of 100 small parts*/
      { recip_max[s]=(recip_max[s]*100/b);
        /*every schd occupies some continue
          parts stored in recip_max array*/
/*
        printf(" %d ",recip_max[s]);
        printf(" p1 ");
*/
      }
/*    *
   c=65536/(2*100);
   k=rand()/c;
*/
   k=random()%100;
   if(k>=100)
     k=99;
   b=0;
   for(oper_no=0;oper_no<6;oper_no++)
      {
/*
        printf(" p2 ");
*/
        b=b+recip_max[oper_no];
        if(b > k)
           break;
      }
   if(oper_no>=6)
      oper_no=5;
/*
   printf("%2d",oper_no);
*/
}
catch_bug()
{ int s,a,b;
  for(s=0;s<schd_no;s++)
    for(a=0;a<strg_no;a++)
      for(b=0;b<max_tk;b++)
        if(bh_order[s][a][b]==non_bhtk)
          { if(b!=max_tk-1)
               { if(bh_order[s][a][b+1]!=non_bhtk)
                   { show_colbh();
                     printf("s=%d,a=%d,b=%d,bug",s,a,b);
                     exit(0);
                   }
               }
          }
}
mutate6()/*exchange normal bhtks in different strings*/
{
  int tt_nom[strg_no];
  int tt_col[strg_no],stop_fg[strg_no],a,b,s,i,j,r;
  unsigned char coll_pos[strg_no][340],coll_no[strg_no][170];
  unsigned char noml_pos[strg_no][340],noml_no[strg_no][170];
  int k0,k1,v0,v1,c,d,k,a0,a1,a2,max;
/*
    for(s=0;s<schd_no;s++)
        for(a=0;a<strg_no;a++)
          {printf("\n");
           for(b=0;b<max_tk;b++)
             printf("%2d",col_bh[s][a][b]);
```

```
          }
*/
   for(s=0;s<schd_no;s++)
     { for(a=0;a<strg_no;a++)/*clear array*/
         {for(b=0;b<max_tk;b++)
            { coll_no[a][b]=non_bhtk;noml_no[a][b]=non_bhtk;
            }
          for(b=0;b<(2*max_tk);b++)
            {coll_pos[a][b]=non_bhtk;noml_pos[a][b]=non_bhtk;
            }
          tt_col[a]=0;stop_fg[a]=0;tt_nom[a]=0;
         }
       for(a=0;a<strg_no;a++)
         { i=0;j=0;
           for(b=0;b<max_tk;b++)/*count the no. of
                                   collided bh in every
                                   strg*/
            { if(col_bh[s][a][b]!=non_bhtk)
                { coll_pos[a][i]=a;coll_pos[a][++i]=b;
                  coll_no[a][((i-1)/2)]=col_bh[s][a][b];
                  ++i;tt_col[a]=tt_col[a]+1;
                }
              else/*col_bh[s][a][b]==non_bhtk*/
                { if(bh_order[s][a][b]!=non_bhtk)
                  /*count the no. of normal bh*/
                   { noml_pos[a][j]=a;noml_pos[a][++j]=b;
                     noml_no[a][((j-1)/2)]=
                     bh_order[s][a][b];++j;
                     tt_nom[a]=tt_nom[a]+1;
                   }
                }
            }
         }
/*
       if(times_no>=4)
         {
           printf("\ns=%d",s);
           for(a=0;a<strg_no;a++)
           { printf("\nnom");
             for(b=0;b<max_tk;b++)
               printf("%2d,",noml_no[a][a]);
             printf("\ncol");
             for(b=0;b<max_tk;b++)
               printf("%2d,",coll_no[a][b]);
           }
           exit(0);
         }
*/
       for(a=0;a<strg_no;a++)
         { if(tt_nom[a]==0)
             stop_fg[a]=1;
         }
       d=0;
       for(a=0;a<strg_no;a++)
         { if(stop_fg[a]==1)
              ++d;
         }
       if(d >= strg_no-1)/*normal strgs are at least two,
                            or do next schd.*/
          { continue;
          }
       max=0;a0=20;/*20 is a dummy value*/
       for(a=0;a<strg_no;a++)/*find two normal strgs with
                               longer f.t*/
         if((str_tm2[s][a]>max) && (stop_fg[a]==0))
           {max=str_tm2[s][a];a0=a;}
```

```
        stop_fg[a0]=1;
        max=0;a1=20;
        for(a=0;a<strg_no;a++)
          if((str_tm2[s][a]>max) && (stop_fg[a]==0))
            {max=str_tm2[s][a];a1=a;}
        r=0;/*find bhtks from different strg and exchange*/
re_find:
        k0=random()%tt_nom[a0];
        v0=noml_no[a0][k0];
        if(k0>=tt_nom[a0])
           {k0=tt_nom[a0]-1;v0=noml_no[a0][k0];}
        k1=random()%tt_nom[a1];
        v1=noml_no[a1][k1];
/*
        if(times_no>=4)
          {printf("\nnom_no\n");
           for(a=0;a<max_tk;a++)
           printf("%2d,",noml_no[a1][a]);
           exit(0);
          }
*/
        if(k1>=tt_nom[a1])
           {k1=tt_nom[a1]-1;v1=noml_no[a1][k1];}
        if(v0==v1)
          { ++r;
            if(r>10)
              break;
            goto re_find;
          }
/*
        if(times_no>=4)
          printf(" v0=%d,v1=%d,k0=%d,k1=%d,a0=%d,a1=%d,
                   d=%d",v0,v1,k0,k1,a0,a1,d);
*/
        bh_order[s][(noml_pos[a0][k0*2])][(noml_pos[a0]
        [k0*2+1])]=v1;
        bh_order[s][(noml_pos[a1][k1*2])][(noml_pos[a1]
        [k1*2+1])]=v0;
      }
}
 /*change normal bhtk with similar bhtk(ex:t00 -> t01 or
 t11 -> t10)*/
mutate5()
{ unsigned char col_pos[340],col_no[170],nom_pos[340],
      nom_no[170];
  int i,j,tt_col=0,tt_nom,s,a,b,d;
  int c,v,e,f,k;
/*
  if(times_no>=82)
  {
    printf("\n mu5 col");
    for(s=0;s<schd_no;s++)
      for(a=0;a<strg_no;a++)
      {printf("\n");
       for(b=0;b<max_tk;b++)
         printf("%2d",col_bh[s][a][b]);
      }
    printf("\n mu5 bh");
    for(s=0;s<schd_no;s++)
      for(a=0;a<strg_no;a++)
      {printf("\n");
       for(b=0;b<max_tk;b++)
         printf("%2d",bh_order[s][a][b]);
      }
  }
*/
```

```
    for(s=0;s<schd_no;s++)
      { for(a=0;a<strg_no;a++)/*clear array*/
          { for(i=0;i<max_tk;i++)
              { col_no[i]=non_bhtk;nom_no[i]=non_bhtk;
              }
            for(i=0;i<(2*max_tk);i++)
              { col_pos[i]=non_bhtk;nom_pos[i]=non_bhtk;
              }
            i=0;tt_col=0;j=0;tt_nom=0;
            for(b=0;b<max_tk;b++)
             { if(col_bh[s][a][b]!=non_bhtk)
               /*count the no of collided bhtks in every strg*/
                { col_pos[i]=a;col_pos[++i]=b;
                  col_no[((i-1)/2)]=col_bh[s][a][b];
                  ++i;++tt_col;
                }
               else/*col_bh[s][a][b]==non_bhtk*/
                { if(bh_order[s][a][b]!=non_bhtk)
                    /*count the no of normal bhtks in every
                      strg*/
                    { nom_pos[j]=a;nom_pos[++j]=b;
                      nom_no[((j-1)/2)]=bh_order[s][a][b];
                      ++j;++tt_nom;
                    }
                }
             }
            if(tt_nom == 0)/*if there are no normal bhtk,
                              skip*/
              goto continu_run;
            k=random()%tt_nom;v=nom_no[k];
           /*replace collided bhtk with the similar bhtk*/
            if(k>=tt_nom)
               {k=tt_nom-1;v=nom_no[k];}
            if(v >= 20)/*bhtk belongs to t2*/
             { for(d=0;d<10;d++)
                 {
                   f=random()%bh_kind[2];
                   if(f>=bh_kind[2])
                      {f=bh_kind[2]-1;}
                   f=f+20;
                   if(v != f)
                     {
/*
                       printf(" col=%d ",
                       col_bh[s][(col_pos[(k*2)])]
                       [(col_pos[(k*2+1)])]);
*/
                       bh_order[s][(nom_pos[(k*2)])]
                       [(nom_pos[(k*2+1)])]=f;
/*
                       printf("v=%d,f=%d,k=%d,p=%d,%d",
                              v,f,k,col_pos[k*2],
                                col_pos[k*2+1]);
*/
                       break;
                     }
                 }
             }
           if((20 > v)  && (v >= 10))/*bhtk belongs to t1*/
             { for(d=0;d<10;d++)
                 {
                   f=random()%bh_kind[1];
                   if(f>=bh_kind[1])
                      f=bh_kind[1]-1;
                   f=f+10;
                   if(v != f)
```

```
                      {
/*
                        printf(" col=%d ",
                        col_bh[s][(col_pos[(k*2)])]
                        [(col_pos[(k*2+1)])]);
*/
                        bh_order[s][(nom_pos[(k*2)])]
                        [(nom_pos[(k*2+1)])]=f;
/*
                        printf(",v=%d,f=%d,k=%d,p=%d,%d",
                                v,f,k,col_pos[k*2],
                                col_pos[k*2+1]);
*/
                        break;
                      }
                  }
              }
          if((10 > v) && (v >= 00))/*bhtk belongs to t0*/
            { for(d=0;d<10;d++)
                {
                  f=random()%bh_kind[0];
                  if(f>=bh_kind[0])
                     f=bh_kind[0]-1;
                  f=f+0;
                  if(v != f)
                    {
/*
                        printf(" col=%d ",
                        col_bh[s][(col_pos[(k*2)])]
                        [(col_pos[(k*2+1)])]);
*/
                        bh_order[s][(nom_pos[(k*2)])]
                        [(nom_pos[(k*2+1)])]=f;
/*
                        printf("v=%d,f=%d,k=%d,p=%d,%d",
                                v,f,k,col_pos[(k*2)],
                                col_pos[(k*2+1)]);
*/
                        break;
                    }
                }
            }
continu_run:;
         }
      }
}
/*exchange collided bhtks and no collided bhtks
  in the same string
*/
mutate4()
{
unsigned char col_pos4[strg_no][340];
unsigned char col_no4[strg_no][170];
unsigned char nom_pos4[strg_no][340];
unsigned char nom_no4[strg_no][170];
int tt_col4[strg_no],tt_nom4[strg_no];
int kk0,kk1,vv0,vv1,rr;
/*
    printf("\n mut4 old col");
    for(s=0;s<schd_no;s++)
      for(a=0;a<strg_no;a++)
        { printf("\n");
          for(b=0;b<max_tk;b++)
            printf("%2d",col_bh[s][a][b]);
        }
    printf("\n mut4 old bh");
```

```
    for(s=0;s<schd_no;s++)
      for(a=0;a<strg_no;a++)
        { printf("\n");
          for(b=0;b<max_tk;b++)
            printf("%2d",bh_order[s][a][b]);
        }
    printf("\n");
*/
  int a,b,s,i,j;
  int c,aa,ss;
   for(s=0;s<schd_no;s++)
     {  for(a=0;a<strg_no;a++)
          {for(b=0;b<max_tk;b++)
             { col_no4[a][b]=non_bhtk;
               nom_no4[a][b]=non_bhtk;
             }
           for(b=0;b<(2*max_tk);b++)
             { col_pos4[a][b]=non_bhtk;
               nom_pos4[a][b]=non_bhtk;
             }
           tt_col4[a]=0;tt_nom4[a]=0;
          }
        for(a=0;a<strg_no;a++)
          { i=0;j=0;
            for(b=0;b<max_tk;b++)
             { if(col_bh[s][a][b]!=non_bhtk)
                 { col_pos4[a][i]=a;col_pos4[a][++i]=b;
                   col_no4[a][((i-1)/2)]=col_bh[s][a][b];
                   ++i;tt_col4[a]=tt_col4[a]+1;
                 }
               else
                 { if(bh_order[s][a][b]!=non_bhtk)
                     { nom_pos4[a][j]=a;nom_pos4[a][++j]=b;
                       nom_no4[a][((j-1)/2)]=
                       bh_order[s][a][b];++j;
                       tt_nom4[a]=tt_nom4[a]+1;
                     }
                 }
             }
          }
        for(a=0;a<strg_no;a++)
        rr=0;
        for(a=0;a<strg_no;a++)
          { if((tt_col4[a]==0) || (tt_nom4[a]==0))
                continue;
 re_find:
              kk0=random()%tt_col4[a];
              vv0=col_no4[a][kk0];
              if(kk0>=tt_col4[a])
                 {kk0=tt_col4[a]-1;vv0=col_no4[a][kk0];}
              kk1=random()%tt_nom4[a];
              vv1=nom_no4[a][kk1];
              if(kk1>=tt_nom4[a])
                 {kk1=tt_nom4[a]-1;vv1=nom_no4[a][kk1];}
              if(vv0==vv1)
                { ++rr;if(rr>10)
                        continue;
                    goto re_find;
                }
              bh_order[s][(col_pos4[a][kk0*2])]
              [(col_pos4[a][kk0*2+1])]=vv1;
              bh_order[s][(nom_pos4[a][kk1*2])]
              [(nom_pos4[a][kk1*2+1])]=vv0;
  /*
              printf("\nmut4 nom and col");
              printf("kk0=%d,kk1=%d,vv0=%d,vv1=%d",kk0,kk1,vv0,vv1);
```

```
              printf("bh0=%d",
              bh_order[s][(col_pos4[a][kk0*2])]
              [(col_pos4[a][kk0*2+1])]);
              printf("bh1=%d",
              bh_order[s][(nom_pos4[a][kk1*2])]
              [(nom_pos4[a][kk1*2+1])]);
*/
          }
     }
/*
   printf("\n new bh");
   for(s=0;s<schd_no;s++)
     for(a=0;a<strg_no;a++)
       { printf("\n");
         for(b=0;b<max_tk;b++)
           printf("%2d",bh_order[s][a][b]);
       }
*/
}
/*new_sch[schd_no],schtm_order[schd_no][2]*/
modify_schedule()
{
   int s,a,b;
   int temp_order[schd_no][strg_no][max_tk];
   int col_bh2[schd_no][strg_no][max_tk];
   if(best_schd!=0)
     { if(best_schd >= schtm_order[0][0])
       /*the f.t of best schd is longer than
         that of new generated best schd*/
         {
/*
              printf("\ng0=%d, sch=%d ",best_schd,
              schtm_order[0][0]);
*/
              best_schd=schtm_order[0][0];
              /*update best schd*/
                   { for(a=0;a<strg_no;a++)
                     /*update the best collides bhtks and
                       the arrangement of best bhtks*/
                       {for(b=0;b<max_tk;b++)
                        {best_col_bh[a][b]=
                         col_bh[(schtm_order[0][1])][a][b];
                         best_bh[a][b]=
                         bh_order[(schtm_order[0][1])][a][b];
                        }
                       best_strtm[a]=
                       str_tm2[(schtm_order[0][1])][a];
                       /*update the f.t's of the best
                         strings*/
                       }
                   }
         }
       else/*best_schd < schtm_order[0][0]*/
         {
/*
              printf("\g0=%d, sch=%d ",best_schd,
                     schtm_order[0][0]);
*/
              for(a=0;a<strg_no;a++)
                 {
                  for(b=0;b<max_tk;b++)
                  /*update new generated best schd with old
                    best schd*/
                    { col_bh[(schtm_order[0][1])][a][b]=
                      best_col_bh[a][b];
                      bh_order[(schtm_order[0][1])][a][b]=
```

```
                      best_bh[a][b];
                    }
                  str_tm2[(schtm_order[0][1])][a]=
                  best_strtm[a];
                 }
         }
     }
   else/*best_schd=0;initilize best schd with the first
        new generated best schd*/
     { best_schd=schtm_order[0][0];
       for(a=0;a<strg_no;a++)
          {for(b=0;b<max_tk;b++)
             { best_col_bh[a][b]=
               col_bh[(schtm_order[0][1])][a][b];
               best_bh[a][b]=
               bh_order[(schtm_order[0][1])][a][b];
             }
           best_strtm[a]=str_tm2[(schtm_order[0][1])][a];
          }
     }
/*
   printf("\n best bh");
   for(a=0;a<strg_no;a++)
     { printf("\n");
       for(b=0;b<max_tk;b++)
         printf("%2d",best_bh[a][b]);
     }
   printf("\n");
*/
   for(s=0;s<schd_no;s++)/*move temporarily the data of
                            old schds into the other
                            arrays*/
     for(a=0;a<strg_no;a++)
        {for(b=0;b<max_tk;b++)
           { temp_order[s][a][b]=bh_order[s][a][b];
             col_bh2[s][a][b]=col_bh[s][a][b];
           }
         str_tm[s][a]=str_tm2[s][a];
        }
   for(s=0;s<schd_no;s++)/*according to the order the new
                            schds,reload the data into
                            bh_order,col_bh and str_tm2
                            arrays*/
     { for(a=0;a<strg_no;a++)
          { for(b=0;b<max_tk;b++)
               { bh_order[s][a][b]=
                 temp_order[(new_sch[s])][a][b];
                 col_bh[s][a][b]=
                 col_bh2[(new_sch[s])][a][b];
                 str_tm2[s][a]=str_tm[(new_sch[s])][a];
               }
          }
     }
   for(s=0;s<schd_no;s++)/*clear str_tm arrays*/
     { old_sch[s]=new_sch[s];
       for(a=0;a<strg_no;a++)
           str_tm[s][a]=0;
     }
/*
   printf("\nnew bh_order");
   for(s=0;s<schd_no;s++)
     for(a=0;a<strg_no;a++)
        {printf("\n");
         for(b=0;b<max_tk;b++)
           printf("%2d",bh_order[s][a][b]);
        }
```

```
   printf("\n new col");
   for(s=0;s<schd_no;s++)
     for(a=0;a<strg_no;a++)
       {printf("\n");
        for(b=0;b<max_tk;b++)
          printf("%2d",col_bh[s][a][b]);
       }
*/
}
/*replace the worst schd with the best schd*/
modify_reproduct()
{  int s,temp,a;
/*
   printf("\nold new_sch");
   for(s=0;s<schd_no;s++)
     printf(" %d",new_sch[s]);
   printf("\t");
*/
   temp=0;a=schd_no-1;/*schd "a" is the worst one*/
find_mintm:
   for(s=0;s<schd_no;s++)
     { if(new_sch[s]==schtm_order[a][1])
       /*find the worst schd from all schds*/
         {temp=1;break;
         }
     }
   if(temp!=1)
     { --a;/*if no worst schd exists,go on finding the
             worse schd */
       if(a!=0)
          goto find_mintm;
       else/*if all schds are best,skip*/
          goto not_change;
     }
   new_sch[s]=schtm_order[0][1];
   /*replace the worst schd in all new
     generated schds with best schd*/
not_change:;
/*
   printf("\nnew new_sch");
   for(s=0;s<schd_no;s++)
     printf(" %d",new_sch[s]);
*/
}
/*max_schtm[s][a]*/
/*arrange the order of schds according to the f.t of
every schd*/
arrang_schtm_order()
{  int s,b,a,m;
   int mintm=0;
   for(s=0;s<schd_no;s++)
     for(a=0;a<2;a++)
       { schtm_order[s][a]=max_schtm[s][a];
         /*max_schtm store the longest f.t in every schd*/
       }
   a=0;
   for(s=0;s<schd_no;s++)
     { mintm=30000;
       for(b=0;b<schd_no;b++)
         { if(schtm_order[b][0] <= mintm)
           /*find a schd with shortest f.t*/
             { mintm=schtm_order[b][0];m=b;
             }
         }
      schtm_order(m][0]=30000;schtm_order[s][1]=m;
      /*store the schd and find next shorter schd*/
```

```
     }
   for(s=0;s<schd_no;s++)
     schtm_order[s][0]=max_schtm[(schtm_order[s][1])][0];
       /*stroe the f.t of all schds*/
/*
   printf("\nschtm_order");
   for(s=0;s<schd_no;s++)
     for(a=0;a<2;a++)
       printf(" %d",schtm_order[s][a]);
*/
}
/*max_schtm[s][a]*/
/*pass the good schds(good genes) to next research nodes*/
reproduction()
{ int s,d;
static float recip_max[schd_no],b=0,c,k;
   for(s=0;s<schd_no;s++)/*add the reciprocals of all
                            f.t's of every schd in a
                            research node*/
      { recip_max[s]=1.0/max_schtm[s][0];
        b=b+recip_max[s];
/*
        printf(" %5.5f,b=%5.5f  ",recip_max[s],b);
*/
      }
   for(s=0;s<schd_no;s++)/*bulid a biased roulette wheel
                            consisting of 100 small parts
                            */
      { recip_max[s]=(recip_max[s]/b)*100;
       /*every schd occupies some continue parts stored
         in recip_max array*/
/*
        printf(" %5.2f ",recip_max[s]);
*/
      }
/*
   c=65536/(2*100.0);
*/
   for(s=0;s<schd_no;s++)
   /*randomly generate a number between 0 and 100 and
     select a new schd according to the number*/
     {
/*
        k=rand()/c;
*/
        k=random()%100;
        if(k>=100)
            k=99;
        b=0;
        for(d=0;d<schd_no;d++)
            { b=b+recip_max[d]+0.0;
              if(b > k)
                {new_sch[s]=d;old_sch[s]=d;break;
                 /*put the new generated schd into new_sch
                   array*/
                }
            }
/*
        printf("\n n=%d,k=%3.1f,b=%3.1f",new_sch[s],k,b);
*/
     }
}
mutate3()/*exchange collided bhtks in the same string*/
{ unsigned char col_pos[strg_no][340],col_no[strg_no][170];
  int tt0_col,tt1_col,a,b,s,i,p,tt_col[strg_no];
  int k0,k1,v0,v1,c;
```

```
/*
   for(s=0;s<schd_no;s++)
     for(a=0;a<strg_no;a++)
       { printf("\n");
         for(b=0;b<30;b++)
           printf("%2d",col_bh[s][a][b]);
       }
   printf("\n");
*/
/*
   for(s=0;s<schd_no;s++)
     for(a=0;a<strg_no;a++)
       { printf("\n");
         for(b=0;b<max_tk;b++)
           printf("%2d",bh_order[s][a][b]);
       }
*/
   for(s=0;s<schd_no;s++)
     { for(a=0;a<strg_no;a++)/*clear arrays*/
         {for(b=0;b<max_tk;b++)
            { col_no[a][b]=non_bhtk;
            }
          for(b=0;b<(2*max_tk);b++)
            { col_pos[a][b]=non_bhtk;
            }
          tt_col[a]=0;
         }
       for(a=0;a<strg_no;a++)/*count the no of collided
                                bhtks in every string*/
         { i=0;
           for(b=0;b<max_tk;b++)
            { if(col_bh[s][a][b]!=non_bhtk)
                { col_pos[a][i]=a;col_pos[a][++i]=b;
                  col_no[a][((i-1)/2)]=col_bh[s][a][b];
                  ++i;tt_col[a]=tt_col[a]+1;
                }
            }
         }
/*
       if(times_no>=82)
          printf("tt_col[0]=%d,tt_col[1]=%d ",tt_col[0],
                  tt_col[1]);
*/
       r=0;
       for(a=0;a<strg_no;a++)
       /*if the no of collided bhtks is 1 of 0,skip*/
         { if(tt_col[a] <= 1)
               continue;
re_find:
             k0=random()%tt_col[a];
             v0=col_no[a][k0];
             /*find two collided*/
             if(k0>=tt_col[a])
                {k0=tt_col[a]-1;v0=col_no[a][k0];}
             k1=random()%tt_col[a];
             v1=col_no[a][k1];
             if(k1>=tt_col[a])
                {k1=tt_col[a]-1;v1=col_no[a][k1];}
/*
             if(times_no>=82)
              printf("\nk0=%d,k1=%d,v0=%d,v1=%d,a=%d,s=%d",
                      k0,k1,v0,v1,a,s);
*/
             if(v0==v1)/*different bhtks*/
               { ++r;
                 if(r>10)
                   goto exit_find;
                 goto re_find;
               }
             bh_order[s][(col_pos[a][k0*2])]
             [(col_pos[a][k0*2+1])]=v1;/*exchange*/
             bh_order[s][(col_pos[a][k1*2])]
             [(col_pos[a][k1*2+1])]=v0;
exit_find:;
         }
     }
/*
   printf("\n");
   for(s=0;s<schd_no;s++)
     for(a=0;a<strg_no;a++)
       { printf("\n");
         for(b=0;b<30;b++)
           printf("%2d",bh_order[s][a][b]);
       }
*/
}
mutate2()/*exchange collided bhtks in different strings*/
{ unsigned char col_pos[strg_no][340];
  char col_no[strg_no][170];
  int tt_col[strg_no],stop_fg[strg_no],a,b,s,i,r;
  int k0,k1,v0,v1,c,d,k,a0,a1,a2,max;
/*
   for(s=0;s<schd_no;s++)
      for(a=0;a<strg_no;a++)
        {printf("\n");
         for(b=0;b<max_tk;b++)
           printf("%2d",col_bh[s][a][b]);
        }
   printf("\n");
*/
/*
   for(s=0;s<schd_no;s++)
      for(a=0;a<strg_no;a++)
        {printf("\n");
         for(b=0;b<max_tk;b++)
           printf("%2d",bh_order[s][a][b]);
        }
*/
   for(s=0;s<schd_no;s++)
     { for(a=0;a<strg_no;a++)/*clear arrays*/
         {for(b=0;b<max_tk;b++)
            { col_no[a][b]=non_bhtk;
            }
          for(b=0;b<(2*max_tk);b++)
            { col_pos[a][b]=non_bhtk;
            }
          tt_col[a]=0;stop_fg[a]=0;
         }
       for(a=0;a<strg_no;a++)
       /*count the no of collided bhtks in every strg*/
         { i=0;
           for(b=0;b<max_tk;b++)
            { if(col_bh[s][a][b]!=non_bhtk)
                { col_pos[a][i]=a;col_pos[a][++i]=b;
                  col_no[a][((i-1)/2)]=col_bh[s][a][b];
                  ++i;tt_col[a]=tt_col[a]+1;
                }
            }
         }
       for(a=0;a<strg_no;a++)
         { if(tt_col[a]==0)
               stop_fg[a]=1;
```

```
          }
        d=0;
        for(a=0;a<strg_no;a++)
          { if(stop_fg[a]==1)
               ++d;
          }
        if(d>=strg_no-1)
        /*if the no of collided strgs is 1 or 0,skip*/
           { continue;
           }
        max=0;a0=20;
        for(a=0;a<strg_no;a++)
        /*find two strings with longer f.t*/
          if((str_tm2[s][a]>max) && (stop_fg[a]==0))
            {max=str_tm2[s][a];a0=a;}
        stop_fg[a0]=1;
        max=0;a1=20;
        for(a=0;a<strg_no;a++)
          if((str_tm2[s][a]>max) && (stop_fg[a]==0))
            {max=str_tm2[s][a];a1=a;}
        r=0;
re_find:
        k0=random()%tt_col[a0];
        v0=col_no[a0][k0];/*find two bhtks*/
        if(k0>=tt_col[a0])
           {k0=tt_col[a0]-1;v0=col_no[a0][k0];}
        k1=random()%tt_col[a1];
        v1=col_no[a1][k1];
        if(k1>=tt_col[a1])
           {k1=tt_col[a1]-1;v1=col_no[a1][k1];}
        if(v0==v1)/*two different bhtks*/
          { ++r;
            if(r>10)
               break;
            goto re_find;
          }
        bh_order[s][(col_pos[a0][k0*2])]
        [(col_pos[a0][k0*2+1])]=v1;/*exchange*/
        bh_order[s][(col_pos[a1][k1*2])]
        [(col_pos[a1][k1*2+1])]=v0;
     }
/*
   printf("\n");
   for(s=0;s<schd_no;s++)
        for(a=0;a<strg_no;a++)
          {printf("\n");
           for(b=0;b<max_tk;b++)
             printf("%2d",bh_order[s][a][b]);
          }
*/
}
/*change collided bhtk with similar bhtk(ex:t00 -> t01 or
t11 -> t10)*/
mutate1()
{ unsigned char col_pos[340];
  char col_no[170];
  int i,tt_col=0,s,a,b,d;
  int c,v,e,f,k;
/*
  printf(" mu1");
  for(s=0;s<schd_no;s++)
    for(a=0;a<strg_no;a++)
      {printf("\n");
       for(b=0;b<max_tk;b++)
         printf("%2d",bh_order[s][a][b]);
      }
```

```
*/
   for(s=0;s<schd_no;s++)
     { for(a=0;a<strg_no;a++)/*clear arrays*/
        { for(i=0;i<max_tk;i++)
            { col_no[i]=non_bhtk;
            }
          for(i=0;i<(2*max_tk);i++)
            { col_pos[i]=non_bhtk;
            }
          i=0;tt_col=0;
          for(b=0;b<max_tk;b++)
          /*count the no of bhtks in every strg*/
           { if(col_bh[s][a][b]!=non_bhtk)
               { col_pos[i]=a;col_pos[++i]=b;
                 col_no[((i-1)/2)]=col_bh[s][a][b];
                 ++i;++tt_col;
               }
           }
          if(tt_col == 0)
          /*if no collided bhtk in every strg,skip*/
            goto continu_run;
          k=random()%tt_col;
          if(k>=tt_col)
             {k=tt_col-1;}
          v=col_no[k];
          /*find a bhtk and replace with the similar
            bhtk*/
          if(v >= 20)/*bhtk belongs to t2*/
           { for(d=0;d<10;d++)
                {
                  f=random()%bh_kind[2];
                  if(f>=bh_kind[2])
                     f=bh_kind[2]-1;
                  f=f+20;
                  if(v != f)
                     {
/*
                       printf(" col=%d ",
                       col_bh[s][(col_pos[(k*2)])]
                       [(col_pos[(k*2+1)])]);
*/
                       bh_order[s][(col_pos[(k*2)])]
                       [(col_pos[(k*2+1)])]=f;
/*
                       printf("v=%d,f=%d,k=%d,p=%d,%d",
                               v,f,k,col_pos[k*2],
                               col_pos[k*2+1]);
*/
                       break;
                     }
                }
           }
         if((20 > v)  && (v >= 10))/*bhtk belongs to t1*/
           { for(d=0;d<10;d++)
                {
                  f=random()%bh_kind[1];
                  if(f>=bh_kind[1])
                     f=bh_kind[1]-1;
                  f=f+10;
                  if(v != f)
                     {
/*
                       printf(" col=%d ",
                       col_bh[s][(col_pos[(k*2)])]
                       [(col_pos[(k*2+1)])]);
*/
```

```
                        bh_order[s][(col_pos[(k*2)])]
                        [(col_pos[(k*2+1)])]=f;
/*
                        printf("v=%d,f=%d,k=%d,p=%d,%d",
                                v,f,k,col_pos[k*2],
                                col_pos[k*2+1]);
*/
                        break;
                      }
                  }
              }
          if((10 > v) && (v >= 00))/*bhtk belongs to t0*/
            { for(d=0;d<10;d++)
                {
                  f=random()%bh_kind[0];
                  if(f>=bh_kind[0])
                     f=bh_kind[0]-1;
                  f=f+0;
                  if(v != f)
                    {
/*
                        printf(" col=%d ",
                        col_bh[s][(col_pos[(k*2)])]
                        [(col_pos[(k*2+1)])]);
*/
                        bh_order[s][(col_pos[(k*2)])]
                        [(col_pos[(k*2+1)])]=f;
/*
                        printf("v=%d,f=%d,k=%d,p=%d,%d",
                                v,f,k,col_pos[(k*2)],
                                col_pos[(k*2+1)]);
*/
                        break;
                      }
                  }
              }
continu_run:;
        }
      }
/*
      printf(" new");
      for(s=0;s<schd_no;s++)
        for(a=0;a<strg_no;a++)
          {printf("\n");
           for(b=0;b<max_tk;b++)
             printf("%2d",bh_order[s][a][b]);
          }
*/
}
/*show bhtks and collided bhtks in every string */
show_bhorder()
{
int a,b,s;
   printf("\n show");
   for(s=0;s<schd_no;s++)
      {
        for(a=0;a<strg_no;a++)
          { printf("\n");
            for(b=0;b<max_tk;b++)
              printf("%2d",bh_order[s][a][b]);
          }
      }
   printf("\n");
}
show_colbh()
{
```

```
int a,b,s;
   printf("\n show");
   for(s=0;s<schd_no;s++)
      {
        for(a=0;a<strg_no;a++)
          { printf("\n");
            for(b=0;b<max_tk;b++)
              printf("%2d",bh_order[s][a][b]);
          }
      }
   printf("\n");
   for(s=0;s<schd_no;s++)
      {
        for(a=0;a<strg_no;a++)
          { printf("\n");
            for(b=0;b<max_tk;b++)
              printf("%2d",col_bh[s][a][b]);
          }
      }
  printf("\n");
}
/*tk[66][5][30],col_bh[5][5][30],bh_order[5][5][30]*/
count_complet_tm()
{
   int s,b,b2,c,c2,a,a2,h=0,h1=0,h2;
   int s1;
   int agv_fg[strg_no];
    /*agv_fg=1:have just counted e.t;agv_fg=2:have just
      counted t,t*/
   int cc[strg_no];/*content pointer in a bhtk*/
   int bb[strg_no];/*bhtk pointer in a strg*/
   int start[strg_no];
    /*the f.t of the ist wkst of a strg is counted,
      its start=1*/
   int start_fg=0;
    /*if the f.t of the 1st wkst of a strg is counted,
      start_fg=start_fg+1*/
   int comp_fg;/*if collision must be happened,comp_fg=1*/
   int comp[strg_no];
    /*if two or more agv's might collide,comp=1*/
   int stop_fg[strg_no];
    /*the f.t of a strg is completed,stop_fg=1*/
   int min,maxstrtm,max,m;
/*
   printf("\n count_complet ");
     printf("\n bh_order");
     for(s=0;s<schd_no;s++)
       for(a=0;a<strg_no;a++)
         { printf("\n");
           for(b=0;b<max_tk;b++)
             printf("%2d",bh_order[s][a][b]);
         }
*/
   for(s=0;s<schd_no;s++)/*clear col_bh array*/
      for(a=0;a<strg_no;a++)
          for(b=0;b<max_tk;b++)
              col_bh[s][a][b]=non_bhtk;
   for(s=0;s<schd_no;s++)
     {
       comp_fg=0;/*inilitialize arrays*/
       for(a=0;a<strg_no;a++)
         { stop_fg[a]=0;agv_fg[a]=2;comp[a]=0;start[a]=0;
           start_fg=0;cc[a]=1;bb[a]=0;
         }
not_end_str:
         for(a=0;a<strg_no;a++)
```

```
    comp[a]=0;
  comp_fg=0;
  min=30000;
  for(a=0;a<strg_no;a++)
    { if(((stop_fg[a]==0) && (str_tm[s][a] < min)) &&
          (start_fg==strg_no))
        { min=str_tm[s][a];a2=a;}
          /*count the f.t of all wkst's after 1st
            wkst of every strg;find min e.t */
      else/*count the f.t of the 1st wkst of every
            strg;find min e.t*/
        { h=bh_order[s][a][(bb[a])];
          if((start[a]!=1) &&
              (tk[h][a][(cc[a])] < min))
            {a2=a;min=tk[h][a][(cc[a])];}
        }
    }
  if(start_fg!=strg_no)
  /*The f.t of 1st wkst in any strg is counted ?*/
    {start[a2]=1;++start_fg;}
  if(min==30000)
  /*if f.t of any strg in a schd is counted,do next
    schd*/
    goto next_schedul;
  b2=bb[a2];c2=cc[a2];
  h=bh_order[s][a2][b2];/*h=bhtk(being counting)*/
  if(agv_fg[a2]==2)
  /*the previous count is to add e.t*/
   { for(a=0;a<strg_no;a++)/*agv will enter a new
                              wkst*/
      { if(a!=a2)
          { h1=bh_order[s][a][(bb[a])];
            if((((agv_fg[a]==1) &&
                  (stop_fg[a]==0)) &&
                  (tk[h1][a][(cc[a]-2)] ==
                   tk[h][a2][c2-1])) &&
                  (str_tm[s][a] > str_tm[s][a2]))
              comp[a]=1;/*might have collision*/
            else
              comp[a]=0;/*no collision*/
          }
      }
     for(a=0;a<strg_no;a++)
      { if(a!=a2)
          { if(comp[a]==1)/*might has collision*/
              {comp_fg=1;break;}
            else
              comp_fg=0;/*no collision*/
          }
      }
     if(comp_fg==1)/*have collision ?*/
      { maxstrtm=0;
        for(a=0;a<strg_no;a++)
          { if(a!=a2)
              { if(comp[a]==1)/*find one strg with
                                longest f.t from
                                collided strgs*/
                  { if(str_tm[s][a] > maxstrtm)
                      maxstrtm=str_tm[s][a];
                  }
              }
          }
        h2=bh_order[s][a2][b2+1];
        /*next bhtk being counting*/
        if(maxstrtm > str_tm[s][a2])
        /*collision is happened,so agv must wait*/
```

```
          { str_tm[s][a2]=maxstrtm+tk[h][a2][c2];
            /*after waiting,add e.t*/
            col_bh[s][a2][b2]=bh_order[s][a2][b2];
            /*store col bhtk*/
          }
        else
          { str_tm[s][a2]=str_tm[s][a2]+
            tk[h][a2][c2];/*no collision*/
          }
        if(tk[h][a2][c2+1]!=0)/*t.t !=0*/
          {agv_fg[a2]=1;cc[a2]=cc[a2]+1;}
        else/*(tk[h][a2][c2+1]==0);t.t=0*/
          { if(tk[h][a2][c2+3]!=0)
            /*next wkst exists*/
              { str_tm[s][a2]=str_tm[s][a2]+
                               tk[h][a2][c2+3];
                agv_fg[a2]=1;cc[a2]=cc[a2]+4;
              }
            else/*(tk[h][a2][c2+3]==0);next wkst
                  doesn't exist*/
              { if((h2!=non_bhtk) &&
                    (tk[h][a2][c2-1]==tk[h2][a2][0]))
                  /*wksts in next bhtk exist;no t.t
                    between old and new bhtks*/
                   { str_tm[s][a2]=str_tm[s][a2]+
                                    tk[h2][a2][1];
                     agv_fg[a2]=1;cc[a2]=2;++b2;
                     bb[a2]=b2;
/*
                     if(times_no==12)
                     printf("\nb2=%d,a2=%d,t=%d\n",
                            --b2,a2,h);
*/
                   }
                  else
                   { if(h2==non_bhtk)
                     /*next bhtk doesn't exist*/
                       { stop_fg[a2]=1;
                         /*stop counting this strg*/
                       }
                     else/*h2!=non_bhtk;
                           tk[h][a2][c2-1]!=
                           tk[h2][a2][0]*/
                       { agv_fg[a2]=1;
                         cc[a2]=cc[a2]+1;
                       }
                   }
              }
          }
      }
    else/*comp_fg=0;no collision;directly add
          executing tm*/
      { h2=bh_order[s][a2][b2+1];/*next bhtk*/
        str_tm[s][a2]=str_tm[s][a2]+tk[h][a2][c2];
        /*add e.t*/
        if(tk[h][a2][c2+1]!=0)/*t.t!=0*/
          { agv_fg[a2]=1;cc[a2]=cc[a2]+1;
            /*pointer to t.t*/
          }
        else/*(tk[h][a2][c2+1]==0);t.t=0*/
          { if(tk[h][a2][c2+3]!=0)
            /*next wkst doesn't exist*/
              { str_tm[s][a2]=str_tm[s][a2]+
                              tk[h][a2][c2+3];
                agv_fg[a2]=1;cc[a2]=cc[a2]+4;
              }
```

```
                    else/*(tk[h][a2][c2+3]==0)*/
                      {if((h2!=non_bhtk) &&
                           (tk[h][a2][c2-1]==tk[h2][a2][0]))
                         { str_tm[s][a2]=str_tm[s][a2]+
                                         tk[h2][a2][1];
                           agv_fg[a2]=1;cc[a2]=2;++b2;
                           bb[a2]=b2;
/*
                           if(times_no==12)
                           printf("\nb2=%d,a2=%d,t=%d\n",b2,a2,h);
*/
                         }
                       else
                         { if(h2==non_bhtk)
                             {stop_fg[a2]=1;}
                           else
                             { agv_fg[a2]=1;
                               cc[a2]=cc[a2]+1;
                             }
                         }
                      }
                  }
              }
          }
        else/*agv_fg[a2]=1;the previous count is to
               add t.t*/
         { str_tm[s][a2]=str_tm[s][a2]+tk[h][a2][c2];
           if(tk[h][a2][c2+2]==0)
             { h2=bh_order[s][a2][b2+1];
               if(h2==non_bhtk)
                 { stop_fg[a2]=1;
                 }
               else
                 { str_tm[s][a2]=str_tm[s][a2]+
                   trav[a2][(tk[h][a2][c2-2])]
                   [(tk[h2][a2][0])];
                   ++b2;bb[a2]=b2;agv_fg[a2]=2;cc[a2]=1;
/*
                   if(times_no==12)
                   printf("\nb2=%d,a2=%d,t=%d\n",b2,a2,h);
*/
                 }
             }
           else/*tk[h][a2][c2+2]!=0;branch task is not yet
                  completed*/
             { agv_fg[a2]=2;cc[a2]=cc[a2]+2;
             }
         }
/*
         if(times_no==12)
           { printf(" %d=%d,",a2,str_tm[s][a2]);
           }
*/
        goto not_end_str;
next_schedul:;
      }
/* reserved
   printf("\nmax_schtm ");
*/
   for(s=0;s<schd_no;s++)
      {max=0;
       for(a=0;a<strg_no;a++)
         { sch_tm[s][a][0]=str_tm[s][a];
           if(str_tm[s][a] > max)
              max=str_tm[s][a];
         }
```

```
       max_schtm[s][0]=max;
/*
       printf(" %5d",max_schtm[s][0]);
*/
      }
/*
   for(s=0;s<schd_no;s++)
      for(a=0;a<strg_no;a++)
        {
          printf("%5d",sch_tm[s][a][0]);
          if(sch_tm[s][a][0]>10000)
            { show_colbh();printf(" higher 10000");
              exit(0);
            }
        }
*/
   for(s=0;s<schd_no;s++)
      for(a=0;a<strg_no;a++)
        str_tm2[s][a]=str_tm[s][a];
/*
   printf("\nbh_order");
   for(s=0;s<schd_no;s++)
      for(a=0;a<strg_no;a++)
        {printf("\n");
         for(b=0;b<max_tk;b++)
           printf("%2d",bh_order[s][a][b]);
        }
   printf("\n");
   for(s=0;s<schd_no;s++)
      for(a=0;a<strg_no;a++)
        {printf("\n");
         for(b=0;b<max_tk;b++)
           printf("%2d",col_bh[s][a][b]);
        }
  printf("\nmax_schtm");
  for(s=0;s<schd_no;s++)
     printf(" %d",max_schtm[s][0]);
*/
}
init_string()
{
   clear_bh_order();
   move_tk();
   move_trav();
   averag_tm();
   arrang_task();
   select_branch();
   inital_tk();
   save_strtm();
}
move_trav()
{
   int a=0,r,c;
/*
   for(c=0;c<30;c++)
     printf("%2d",tk[0][0][c]);
   printf("input move");
*/
   for(r=0;r<wk_kind;r++)
     for(c=0;c<wk_kind;c++)
       trav[a][r][c]=trav0[r][c];
   ++a;
   for(r=0;r<wk_kind;r++)
     for(c=0;c<wk_kind;c++)
       trav[a][r][c]=trav1[r][c];
   ++a;
```

```
    for(r=0;r<wk_kind;r++)
      for(c=0;c<wk_kind;c++)
        trav[a][r][c]=trav2[r][c];
/*
    ++a;
    for(r=0;r<wk_kind;r++)
      for(c=0;c<wk_kind;c++)
        trav[a][r][c]=trav3[r][c];
    ++a;
    for(r=0;r<wk_kind;r++)
      for(c=0;c<wk_kind;c++)
        trav[a][r][c]=trav4[r][c];
*/
/*
    printf("\n");
    for(c=0;c<30;c++)
      printf("%2d",tk[0][0][c]);
    printf("trav");exit(0);
*/
/*
    for(a=0;a<schd_no;a++)
      {printf("\n");
       for(r=0;r<wk_kind;r++)
         {printf("\n");
          for(c=0;c<wk_kind;c++)
            printf("%2d",trav[a][r][c]);
         }
      }
*/
}
save_strtm()
{
    int s,a,b,re;
    for(s=0;s<schd_no;s++)
      { for(a=0;a<strg_no;a++)
          { for(b=0;b<max_tk;b++)
              { re=bh_order[s][a][b];
                if(re!=non_bhtk)
                  sch_tm2[s][a]=sch_tm2[s][a]+
                                bhftsum[a][re];
              }
/*
            printf(" %d",sch_tm2[s][a]);
*/
          }
      }
}
clear_bh_order()
{
    int s,a,b;
    for(s=0;s<schd_no;s++)
        for(a=0;a<strg_no;a++)
             for(b=0;b<max_tk;b++)
                 { bh_order[s][a][b]=non_bhtk;
                   col_bh[s][a][b]=non_bhtk;
                 }
    for(s=0;s<schd_no;s++)
      for(b=0;b<300;b++)
        tk_order[s][b]=non_bhtk;
}
inital_tk()
{
    int a,s,t,e,pass,tm,sum;
/*
    for(s=0;s<schd_no;s++)
       { printf("\n");
```

```
          for(t=0;t<ttkno;t++)
            printf("%2d",tk_order[s][t]);
        }
    printf("\n inital_tk input");
*/
    for(s=0;s<schd_no;s++)
          { a=0;pass=0;sum=0;e=0;
            for(t=0;t<ttkno;t++)
                { tm=bhftsum[a][(tk_order[s][t])];
                  sum=tm+sum;
                  if (pass==0)
                      { if    (sum-meantm < 0)
                              { bh_order[s][a][t-e]=
                                tk_order[s][t];
                              }
                        else {
/*
                              printf("%5d",sum_tm);
*/
                              ++a;
                              sum=0;e=t;--t;
                              if    (a==(strg_no-1))
                                    pass=1;
                              else pass=0;
                             }
                      }
                  else { bh_order[s][a][t-e]=
                         tk_order[s][t];
                       }
                }
/*
                printf("%5d",sum);
*/
          }
/*
        printf("\nbh_order\n");
        for(s=0;s<schd_no;s++)
           { for(a=0;a<strg_no;a++)
                { for(t=0;t<max_tk;t++)
                      printf("%2d",bh_order[s][a][t]);
                  printf("\n");
                }
           }
        exit(0);
*/
}
select_branch()
{
    int b,s,r,k,c;
/*
    for(s=0;s<schd_no;s++)
      { printf("\n");
        for(b=0;b<ttkno;b++)
          printf("%2d",tk_order[s][b]);
      }
    printf("\n select_branch tk");
*/
    for(s=0;s<schd_no;s++)
       { b=0;
anoth_strg:
          c=tk_order[s][b];
          switch(c)
                { case 0:
/*
                            r=65536/2/bh_kind[0];
                            k=rand()/r;
```

```
*/
                        k=random()%bh_kind[0];
                        if(k==0)
                           tk_order[s][b]=0;
                        if(k==1)
                           tk_order[s][b]=1;
                        if(k>1)
                           tk_order[s][b]=0;
                        break;
                 case 1:
/*
                        r=65535/2/bh_kind[1];k=rand()/r;
*/
                        k=random()%bh_kind[1];
                        if(k==0)
                           tk_order[s][b]=10;
                        if(k==1)
                           tk_order[s][b]=11;
                        if(k>1)
                           tk_order[s][b]=10;
                        break;
                 case 2:
/*
                        r=65535/2/bh_kind[2];k=rand()/r;
*/
                        k=random()%bh_kind[2];
                        if(k==0)
                           tk_order[s][b]=20;
                        if(k==1)
                           tk_order[s][b]=21;
                        if(k==2)
                           tk_order[s][b]=22;
                        if(k==3)
                           tk_order[s][b]=23;
                        if(k==4)
                           tk_order[s][b]=24;
                        if(k==5)
                           tk_order[s][b]=25;
                        if(k>5)
                           tk_order[s][b]=20;
                        break;
                 default:printf("select_branch in trouble");break;
               }
          ++b;
          if(b!=ttkno)
            goto anoth_strg;
        }
/*
   for(s=0;s<schd_no;s++)
       { printf("\n");
         for(b=0;b<ttkno;b++)
           printf("%2d",tk_order[s][b]);
       }
   printf("\n select branch");exit(0);
*/
}
move_tk()
{
   int a,b;
   for(a=0;a<agv_no;a++)
       { for(b=0;b<bh_contt;b++)
            { tk[00][a][b]=tk00[a][b];
              tk[01][a][b]=tk01[a][b];
              tk[10][a][b]=tk10[a][b];
              tk[11][a][b]=tk11[a][b];
              tk[20][a][b]=tk20[a][b];
              tk[21][a][b]=tk21[a][b];
              tk[22][a][b]=tk22[a][b];
              tk[23][a][b]=tk23[a][b];
              tk[24][a][b]=tk24[a][b];
              tk[25][a][b]=tk25[a][b];
            }
       }
/*
   printf("\n");
   for(a=0;a<1;a++)
       { for(b=0;b<30;b++)
            printf("%d,",tk[00][a][b]);
       }
   exit(0);
*/
}
arrang_task()
{
   int a,d,e,f,g,h,tt=0,tkno[tk_kind],
       tkno1[tk_kind];/*tk_kind=3*/
   int b,c,s,correct_tkno[schd_no][tk_kind];
   ttkno=0;
   for(a=0;a<tk_kind;a++)/*ehtk_no[a]={40,40,24}*/
      { if (ehtk_no[a] !=0)
           ttkno=ehtk_no[a]+ttkno;/*sum of total tasks*/
        else break;
      }
/*
   c=65536/(2*ttkno);
*/
/*
   printf("tt=%d,c=%1.1f",ttkno,c);
   printf("\n");
*/
   for(a=0;a<tk_kind;a++)
      { tkno[a]=ehtk_no[a];tkno1[a]=ehtk_no[a];}
   d=ttkno;
   for(e=0;e<schd_no;e++)
    {
/*
      printf("\ne=%d",e);
*/
      for(c=0;c<tk_kind;c++)
         correct_tkno[e][c]=0;
      for(a=0;a<tk_kind;a++)
          tkno[a]=tkno1[a];
      for(a=0;a<d;a++)
          {
/*
            b=rand()/c;
*/
            b=random()%ttkno;
            tt=0;
            for(h=0;h<tk_kind;h++)/*select agv*/
               { tt=tt+tkno[h];
                 if    (tt > b)
                       break;
                 else continue;
               }
/*
            printf(" b=%d,tt=%d",b,tt);
*/
            if(h>=tk_kind)/*there is doubt here(h is 3),
                            we correct h=0*/
               h=0;
            tkno[h]=tkno[h]-1;
```

```
            tk_order[e][a]=h;
            /*reduce a task from tkno[h]
              ;add the task to tk_order[e][a] e=sched*/
            ttkno=ttkno-1;/*decrease a task from total
                               tasks*/
            if (ttkno==0)
                tk_order[e][a]=h;
/*
            else c=random()%ttkno;
*/
/*
            else c=65536/(2*ttkno);
*/
/*
            printf("2=%d,1=%d",tkno[2],tkno[1]);
            printf(",0=%d,ttkno=%d,or=%d,c=%d\n",
                   tkno[0],ttkno,tk_order[e][a],c);
*/
            if (ttkno==0)
               break;
         }
/*
         for(f=0;f<d;f++)
            printf("tk_order[e][f]=%2d,",tk_order[e][f]);
         printf("\n");
*/
         ttkno=d;
   }
 /*
   printf("\ntk order\n");
   for(s=0;s<schd_no;s++)
      { printf("\n");
        for(b=0;b<d;b++)
          printf("%2d",tk_order[s][b]);
      }
   printf("\n");
*/
   /*After selecting tks,correct no of tasks*/
   for(s=0;s<schd_no;s++)
      for(b=0;b<d;b++)
        { if(tk_order[s][b]==0)
            {++correct_tkno[s][0];
             if(correct_tkno[s][0]>ehtk_no[0])
               { if(correct_tkno[s][1]<ehtk_no[1])
                   { tk_order[s][b]=1;
                     ++correct_tkno[s][1];
                   }
                 else
                   { tk_order[s][b]=2;
                     ++correct_tkno[s][2];
                   }
                 --correct_tkno[s][0];
               }
             continue;
            }
          if(tk_order[s][b]==1)
            {++correct_tkno[s][1];
             if(correct_tkno[s][1]>ehtk_no[1])
               { if(correct_tkno[s][2]<ehtk_no[2])
                   { tk_order[s][b]=2;
                     ++correct_tkno[s][2];
                   }
                 else
                   { tk_order[s][b]=0;
                     ++correct_tkno[s][0];
                   }
                 --correct_tkno[s][1];
               }
             continue;
            }
          if(tk_order[s][b]==2)
            {++correct_tkno[s][2];
             if(correct_tkno[s][2]>ehtk_no[2])
               { if(correct_tkno[s][0]<ehtk_no[0])
                   { tk_order[s][b]=0;
                     ++correct_tkno[s][0];
                   }
                 else
                   { tk_order[s][b]=1;
                     ++correct_tkno[s][1];
                   }
                 --correct_tkno[s][2];
               }
             continue;
            }
        }
/*
   for(s=0;s<schd_no;s++)
      { printf("\n");
        for(b=0;b<d;b++)
          printf("%2d",tk_order[s][b]);
      }
   printf("\n arrange tk");
   exit(0);
*/
}
/*tk[tb][s][c]=[task branch][agv][content]*/
averag_tm()
{
   int a,tb,c;
   int tt;
/*
   for(c=0;c<bh_contt;c++)
        printf("%2d",tk[00][0][c]);
   exit(0);
*/
   for(tb=0;tb<(tk_kind*10);tb++)
       { tt=0;
         for(c=0;c<bh_contt;c++)
            { tt=tt+tk[tb][0][++c];tt=tt+tk[tb][0][++c];
              tkbhsum[(tb/10)]=tt;
              /*tkbhsum[(tb/10)]=the f.t's of t00,t10,t20
                ,t30,t40*/
              if(tk[tb][0][c+2]==0)
                 break;
            }
         tb=tb+9;
       }
 /*
    printf("\n");
    for(tb=0;tb<tk_kind;tb++)
        printf("%d,",tkbhsum[tb]);
    printf("\n");
    exit(0);
 */
    tt=0;
    for(tb=0;tb<tk_kind;tb++)
        tt=tt+ehtk_no[tb]*tkbhsum[tb];
    meantm=tt/strg_no;
    /*meantm=the rough f.t of every string*/
 /*
    printf("\nmeantm=%d",meantm);
```

```
    exit(0);
*/
}
/*sch_tm2[5][2]*/
modify_init_string()
{
    int i,j;
    int s,a,s1,b,temp,b2,k;
    int ft[strg_no],min_tm,d,e,ft0,ft1,tm0,tm1,tm00,tm11;
    int max,min,a0,a1;
/* reserved
for(a=0;a<strg_no;a++)
   { printf("\n");
     for(b=0;b<max_tk;b++)
       printf("%2d",bh_order[0][a][b]);
   }
printf("\n");
for(s=0;s<schd_no;s++)
   for(a=0;a<strg_no;a++)
     printf("%5d",sch_tm2[s][a]);
*/
for(k=0;k<5;k++)
   {for(s=0;s<schd_no;s++)
      {
/*
        printf("\n");
        for(a=0;a<strg_no;a++)
           {printf("\n");
            for(b=0;b<max_tk;b++)
              printf("%2d",bh_order[s][a][b]);
           }
        exit(0);
*/
        for(a=0;a<strg_no;a++)
          ft[a]=sch_tm2[s][a];
        min=30000;max=0;
        for(a=0;a<strg_no;a++)
           { if(ft[a]>max)
               {max=ft[a];a0=a;}
           }
        for(a=0;a<strg_no;a++)
           { if(ft[a] < min)
               { if(a0!=a)
                    {min=ft[a];a1=a;}
               }
           }
        s=0;b2=0;
        if((ft[a0]-ft[a1]) > 5)
           {
retry:
             for(b=0;b<max_tk;b++)
               {if(bh_order[s][a0][b]!=non_bhtk)
                  {
                    tm0=bhftsum[a0][(bh_order[s][a0][b])];
                    tm00=bhftsum[a1][(bh_order[s][a0][b])];
                    tm1=bhftsum[a1][(bh_order[s][a1][b2])];
                    tm11=bhftsum[a0][(bh_order[s][a1][b2])];
                    if( tm0 > tm1)
                       { if(abs((ft[a0]-tm0+tm11)-
                                (ft[a1]-tm1+tm00)) <
                            abs((ft[a0]-ft[a1])))
                           { temp=bh_order[s][a0][b];
                             bh_order[s][a0][b]=
                             bh_order[s][a1][b2];
                             bh_order[s][a1][b2]=temp;
                             ft[a0]=ft[a0]-tm0+tm11;
                             ft[a1]=ft[a1]-tm1+tm00;
                             sch_tm2[s][a0]=ft[a0];
                             sch_tm2[s][a1]=ft[a1];
/*
                             printf("\n %d %d",
                             sch_tm2[s][a0],sch_tm2[s][a1]);
                             for(i=0;i<2;i++)
                                {printf("\n");
                                 for(j=0;j<max_tk;j++)
                                 printf("%2d",bh_order[s][i][j]);
                                }
*/
                           }
                       }
                    else/*tm0 <= tm1*/
                      {if(ft[a1]>ft[a0])
                         {
                          if(abs((ft[a1]-tm1+tm00)-
                                 (ft[a0]-tm0+tm11)) <
                             abs((ft[a1]-ft[a0])))
                            { temp=bh_order[s][a1][b2];
                              bh_order[s][a1][b2]=
                              bh_order[s][a0][b];
                              bh_order[s][a0][b]=temp;
                              ft[a1]=ft[a1]-tm1+tm00;
                              ft[a0]=ft[a0]-tm0+tm11;
                              sch_tm2[s][a0]=ft[a0];
                              sch_tm2[s][a1]=ft[a1];
/*
                              printf("\n %d %d",
                              sch_tm2[s][a0],sch_tm2[s][a1]);
                              for(i=0;i<2;i++)
                                  { printf("\n");
                                    for(j=0;j<max_tk;j++)
                                       printf("%2d",
                                       bh_order[s][i][j]);
                                  }
*/
                            }
                         }
                      }
                  }
                else/*bh_order[s][a][b]==non_bhtk*/
                  { if(abs(ft[a1]-ft[a0])) > 5)
                      {++b2;
                       if(bh_order[s][a1][b2]!=non_bhtk)
                          goto retry;
                      }
                  }
               }
           }
      }
   }
/*
printf("\ninitial string");
for(s=0;s<schd_no;s++)
   for(a=0;a<strg_no;a++)
      printf("%5d",sch_tm2[s][a]);
printf("\nbh_order");
      for(s=0;s<schd_no;s++)
           for(a=0;a<strg_no;a++)
             { printf("\n");
                for(b=0;b<max_tk;b++)
                   printf("%2d",bh_order[0][a][b]);
             }
      printf("\nmodify string");exit(0);
```