

國立交通大學  
工業工程與管理學系碩士班

碩士論文

應用改善式啟發解與基因演算法求解  
晶圓針測排程問題之最大完成時間最小化

**Improving Heuristics and Genetic Algorithms for  
Minimizing the Maximum Completion Time for the  
Wafer Probing Scheduling Problem (WPSP)**



研究生：蔡育燐

指導教授：彭文理博士

楊明賢博士

中華民國九十三年六月

# 應用改善式啓發解與基因演算法求解 晶圓針測排程問題之最大完成時間最小化

研究生：蔡育燐

指導教授：彭文理 博士

國立交通大學工業工程與管理學系碩士班

## 摘要

晶圓廠針測區排程問題(Wafer Probing Scheduling Problem, WPSP)是平行機台排程問題的實例應用，另外也可應用在積體電路(IC)製造業以及其他的工業用途。求解目標式為最小化機台總工作量之晶圓廠針測區排程問題可能會導致平行機台間負荷的不平衡，而無法被現場監控者所接受。因此，在本篇論文中我們把目標式改為最小化最大完成時間之晶圓廠針測區排程問題並用一整數規劃問題來描述之。為了有效解決晶圓廠針測區排程問題之最大完成時間最小化，我們提出了結合初估產能負荷及晶圓廠針測區排程問題的演算法來做重複的求解之改善式啓發解法。此外，我們還提出了不同於以往的混合式基因演算法，其使用之初始母體為晶圓廠針測區排程問題的演算法所求得的排程解與部分排程保留導向的基因交配法來對問題作排程的流程。為了有效評估兩演算法在不同問題情境下的績效，本論文使用了四種影響晶圓廠針測區排程問題之特性來產生產生多組不同的問題。測試結果發現改善式啓發解在我們所設計的問題下其平行機台排程工件之最大完工時間比混合式基因演算法來的好。且當混合式基因演算法使用改善式啓發解之排程解當作初始母體時，其在某些問題情境下還可對改善式啓發解的排程解做更進一步的改善。

關鍵詞：晶圓針測，平行機台排程，最小化最大完成時間，改善式啓發解，基因演算法

# **Improving Heuristics and the Hybrid Genetic Algorithm for minimizing the maximum completion time of the Wafer Probing Scheduling Problem (WPSP)**

Student : Yu-Lin Tsai

Advisor : Dr. Wen-Lea Pearn

Department of Industrial Engineering and Management

National Chiao Tung University

## **Abstract**

The wafer probing scheduling problem (WPSP) is a practical version of the parallel-machine scheduling problem, which has many real-world applications including the integrated circuit (IC) manufacturing industry and other industries. WPSP carries the objective to minimize the total machine workload, which might lead to unbalanced workloads among the parallel machines and be unaccepted for the shop floor supervisors. Therefore, we consider WPSP with the objective to minimize the maximum completion time and formulate the WPSP with minimum makespan as an integer-programming problem. To solve the WPSP with minimum makespan effectively, we proposed the improving heuristics, which add the expected machine load into savings and insertion algorithms for solving problems repeatedly. Besides, we also provide hybrid GA including initial population by WPSP algorithms and sub-schedule preservation crossover to solve the considered problem. To evaluate the performance of the two proposed approaches under various conditions, the performance comparison on a set of test problems involving four problem characteristics are provided. The computational result shows that improving heuristics are better than hybrid GA in scheduling solutions and velocities of WPSP with minimum makespan. When hybrid GA is using initial population by improving heuristics, it can make further improvement for the best solution of improving heuristics in some situations.

*Keyword:* wafer probing, parallel-machine scheduling, minimum makespan, improving heuristics, genetic algorithms

## Contents

Abstract.....	iii
Contents .....	iv
List of Tables.....	v
List of Figures .....	vi
Notations .....	vii
1. Introduction.....	1
2. Problem definition and formulation.....	2
3. The Improving Heuristics .....	6
3.1 Phase I - Existing network algorithms .....	8
3.2 Phase II - Network Adjusting Procedure .....	14
4. The Genetic Algorithm For WPSP with min $C_{max}$ .....	15
4.1 GA for Parallel Machine Problem.....	16
4.2 The Hybrid Genetic Algorithm .....	21
4.2.1 Problem encoding .....	22
4.2.2 Initialization .....	23
4.2.3 Selection.....	23
4.2.4 Genetic operators.....	24
4.2.5 Elitist Strategy.....	27
4.2.6 Stopping Criteria.....	28
5. Problem Design and Testing .....	28
6. Computational Results .....	32
6.1 ANOVA Analysis of Improving Heuristics.....	32
6.2 Computational Results of GA with Initial Population from WPSP algorithms .....	35
6.3 Further Improvement of Hybrid GA with Initial Population from Improving Heuristics .....	36
7. Conclusion .....	37
Reference .....	39
Appendix.....	43

## List of Tables

Table 1. The comparison of GA under various problem characters, crossover, and mutation. ....	18
Table 2. Summary of 16 problem design .....	31
Table 3. Computational results of WPSP algorithms in 16 test problems. ....	33
Table 4. Computational results of improving heuristics under all kinds of situations. ....	33
Table 5. Check of normality assumption for 96 solutions in 16 test problems.....	34
Table 6. The summary of ANOVA table under 99% confidential intervals.....	34
Table 7. Duncan’s multiple comparisons for the performance of WPSP algorithms... ..	34
Table 8. The comparison of hybrid GA with WPSP algorithms and improving heuristics in testing problems.....	36
Table 9. Computation results of GA with initialization of improving heuristics.....	37
Table A1. Processing times of jobs with product types and product families under low total processing time level.....	43
Table A2. Processing times of jobs with product types and product families under high total processing time level.....	43
Table A3. Tightness of due dates in 16 testing problems.....	43
Table A4. All jobs with product types and due dates while tightness of due dates is stable and total processing time level is low.....	44
Table A5. All jobs with product types and due dates while tightness of due dates is increasing and total processing time level is low. ....	44
Table A6. All jobs with product types and due dates while tightness of due dates is stable and total processing time level is high. ....	44
Table A7. All jobs with product types and due dates while tightness of due dates is increasing and total processing time level is high. ....	45
Table A8. Setup time with product types while product family ratio is 2 and temperature changing is not considered. ....	45
Table A9. Setup time with product types while product family ratio is 2 and temperature changing is considered. ....	46
Table A10. Setup time with product types while product family ratio is 6 and temperature changing is not considered. ....	46
Table A11. Setup time with product types while product family ratio is 6 and temperature changing is considered.....	47

## List of Figures

Figure 1. The structure of the improving heuristic .....	7
Figure 2. The flowchart of the execution for hybrid genetic algorithm.....	22
Figure 3. Copy the partitioning structure to offspring .....	25
Figure 4. Copy the sub-schedules to offspring .....	26
Figure 5. Fill the empty of the offspring from the other parent. ....	26
Figure 6. Illustration of swapping technique .....	27
Figure 7. The trend of average solutions in population of hybrid GA repeated four times in problem No. 7.....	47
Figure 8. The trend of best solution in population of hybrid GA repeated four times in problem No. 7. ....	48
Figure 9. The trend of average solutions in population of hybrid GA repeated four times in problem No. 8.....	48
Figure 10. The trend of average solutions in population of hybrid GA repeated four times in problem No. 8.....	49
Figure 11. The further improvement of hybrid GA with initialization by improving heuristics in problem No. 8.....	49
Figure 12. The further improvement of hybrid GA with initialization by improving heuristics in problem No. 16.....	50

## Notations

### IP modeling

- $R_j$  : the  $j$ th product type of jobs
- $I_j$  : the number of jobs in  $j$ th product type of jobs
- $m_k$  : the  $k$ th machine of identical parallel machines
- $r_i$  : the job of WPSP with minimum makespan
- $F$  : the total number of product families
- $J$  : the total number of product types
- $J_f$  : the total number of product types in product family  $f$
- $R_j$  : the  $j$ th subset (product type) of jobs to be processed
- $Cmax$  : the maximum completion time (makespan)
- $W$  : the predetermined machine capacity expressed in terms of processing time units
- $s_{i'}$  : the sequentially dependent setup time between any two consecutive jobs  $r_i$  and  $r_{i'}$
- $x_{ik}$  : the variable indicating whether the job  $r_i$  is scheduled on machine  $m_k$
- $p_i$  : the processing time of job  $r_i$  in cluster  $R_j$  ( $r_i \in R_j$ )
- $t_{ik}$  : the starting time of job  $r_i$  to be processed on machine  $m_k$
- $b_i$  : the ready time of job  $r_i$
- $d_i$  : the due date of job  $r_i$
- $e_i$  : the latest starting processing time of job  $r_i$
- $y_{i'k}$  : the precedence variable, which should be set to 1 if the two jobs  $r_i$  and  $r_{i'}$  are scheduled on machine  $m_k$  and job  $r_i$  precedes job  $r_{i'}$  (not necessarily directly), and 0 otherwise
- $z_{i'k}$  : the direct-precedence variable, which should be set to 1 if the two jobs  $r_i$  and  $r_{i'}$  are scheduled on machine  $m_k$  and job  $r_i$  precedes job  $r_{i'}$  directly, and 0 otherwise.
- $Q$  : a constant, which is chosen to be sufficiently large enough to make constraints of IP model satisfied

### Improving Heuristics

- $Max[s_{U_i}]$  : the maximal setup time of machine switching from idle status (denoted by the label “U”) to processing status
- $Max[s_{iU}]$  : the maximal setup time of machine switching from processing status to

- idle status
- $Max1[s_{ii'}]$  : the maximal setup time of two consecutive jobs processed on machine coming from different product type and different product family
- $Max2[s_{ii'}]$  : the maximal setup time of two consecutive jobs from different product type and same product family
- $\sigma$  : the parameter which may vary according to the problem data structure used in estimation of expected machine load
- $ES$  : the sum of expected setup time on identical parallel machines
- $EL$  : the expected machine workload
- $i_{vk}$  : the selected job has been scheduled on the  $v$ th position of machine  $m_k$
- $SA_{ii'}$  : the saving value for any pairs of jobs  $r_i$  and  $r_{i'}$ , and U denotes the idle status
- $MSA_{ii'}$  : the modified saving value for any pairs of jobs  $r_i$  and  $r_{i'}$ , and U denotes the idle status
- $A$  : the parameter added into the savings function of modified sequential saving (MSA) to present the percentage of postponement restriction
- $B$  : the parameter added into the savings function of modified sequential saving (MSA) to present the percentage of time window restriction
- $c_1(u, k, v)$  : the insertion cost of job  $u$  added into the  $v$ th position on machine  $m_k$  in sequential insertion (SIA)
- $c'_1(u, k, v^*)$  : the best insertion cost of job  $u$  added into the  $v^*$ th position on machine  $m_k$  in sequential insertion (SIA)
- $c_2(u)$  : the regret value of job  $u$  in sequential insertion (SIA)
- $c'_2(u^*)$  : the largest regret value among all unscheduled jobs in sequential insertion (SIA)
- $c''_1(u, k^*, v^*)$  : the best insertion cost of job  $u$  added into the  $v^*$ th position on machine  $m_{k^*}$  in parallel insertion (PIA)
- $c_3(u)$  : the regret value of job  $u$  in parallel insertion (PIA)
- $c'_3(u^*)$  : the largest regret value of job  $u^*$  among all unscheduled jobs in parallel insertion (PIA)
- $c_{11}(u, k, v)$  : the modified insertion cost of job  $u$  added into the  $v$ th position on machine  $m_k$  in parallel insertion with the slackness (PIA II)
- $\lambda$  : the parameter which represents the ratio of the insertion values that the slackness would have in parallel insertion with the slackness (PIA II)



- $c_4(u)$  : the modified regret value of job  $u$  in parallel insertion with the variance of regret measure (PIA IV)
- $Var(c'_1)$  : the variance of best insertion cost between all parallel machines in parallel insertion with the variance of regret measure (PIA IV)
- $Avg(c'_1)$  : the average value of best insertion cost on all identical parallel machines in parallel insertion with the variance of regret measure (PIA IV)
- $\varphi$  : the parameter which determines the schedule ranking of all jobs on all identical parallel machines in parallel insertion with the variance of regret measure (PIA IV)
- $LB$  : lower bound of the adjusting procedure
- $UB$  : upper bound of the adjusting procedure
- $ML_k$  : the total machine load on machine  $m_k$
- $\delta$  : the repeat times of executing WPSP algorithms in improving heuristics

### Hybrid GA

- $C \max_{\alpha, \beta}$  : the makespan of the  $\alpha$  th chromosome in the pool when the GA cycle proceeds to the  $\beta$  th generation
- $K$  : the available machine number of identical parallel machines
- $F(\alpha, \beta)$  : the fitness value of the  $\alpha$  th chromosome in the pool when the GA cycle proceeds to the  $\beta$  th generation
- $d_{\nu k}$  : the due date of job on  $\nu$  th position of machine  $m_k$
- $SSL_{\nu k}$  : the sum of slackness values for  $n$  consecutive jobs on machine  $m_k$ , which are located from  $\nu$  th to  $(\nu + n - 1)$  th position
- $\overline{SSL}_{\nu k}$  : the average value of slackness for  $n$  jobs started from  $\nu$  th to  $(\nu + n - 1)$  th position on machine  $m_k$
- $n$  : the number of job consideration on machine  $m_k$  while estimating the slackness of job combinations
- $N_k$  : the number of jobs on machine  $m_k$
- $R$  : the product family ratio in testing problems
- $TI(Y)$  : the tightness value of jobs before  $Y$ th due date point
- $P(Y)$  : the total processing time of jobs of which due dates are given before  $Y$ th due date point
- $Cap(Y)$  : the available capacity of machine before  $Y$ th due date point
- $Num(Y)$  : the number of jobs of which due dates are given before  $Y$ th due date point

## 1. Introduction

The wafer probing scheduling problem (WPSP) [1,2] is a practical version of the parallel-machine scheduling problem, which has many real-world applications, especially in the integrated circuit (IC) manufacturing industry. There are wafer fabrication, wafer sorting, assembly, and final test in the processes of IC product, and the first and fourth stages are related processes, where the testers are expensive and critical. Pearn et al. [2] considered the wafer probing scheduling problem (WPSP) with the objective of minimizing total workload. They formulated the WPSP as an integer programming problem and transformed the WPSP into the vehicle routing problem with time window (VRPTW). They provided three VRPTW algorithms for solving the WPSP and their computational results showed that the network transformation of identical parallel machine scheduling was efficient and applicable under the objective of raising identical parallel machine utilities. A minimizing-makespan schedule not only can result in higher efficiency and resource utilization but minimize the time, which jobs are operating in the factory. The identical parallel-machine scheduling problem with minimized makespan also identifies the bottleneck machine, which needs to be arranged carefully.

This paper considers the identical parallel-machine scheduling problem with minimizing makespan, which has been proved to be a NP problem [3,4]. In view of the NP-hard nature of the problem, several polynomial time algorithms have been proposed for its solution. It has been traditional solved by operational methods such as integer programming, branch and bound method, dynamic programming, etc.[5-9]. Min et al. [10] proposed a genetic algorithm, which contains the procedures of coding, initializing, reproducing, crossover, and mutation. It was actually efficient for large scale problems. Besides, Gupta et al. [11] provided a LISTFIT algorithm, or the SPT/LPT and MULTIFIT procedure, for solving parallel-machine scheduling problems with minimizing makespan. Some proposed algorithms also have been used for solving parallel-machine scheduling with other objectives. Azizoglu and Kirca [12] proposed a branch and bound algorithm combined with lower bounding

scheme for the objective of minimizing total tardiness. Lee and pinedo [13] presented a three phases algorithm incorporating the ATCS rule and simulated annealing method for minimizing the sum of weighted tardiness and the experimental results showed that simulated annealing method had a great improvement of solutions. Park, Kim, and Lee [14] addressed an extension of the ATCS (Apparent Tardiness Cost with Setups) rule for the objective of minimizing total weighted tardiness. Hurkens et al. [15] proposed a 0-1 interchange, which is the procedure of the job moving iteratively to the machine with minimal load if its processing time is less than the difference between maximum and minimum machine load. Veen et al. [16] formulated an integer programming by dividing jobs into  $K$  job-classes and considered that the change-over time between two consecutive jobs is dependent on the job-class to which the two jobs belonged.

For the WSPSP with the objective of minimizing makespan, we first formulate our problem as an integer programming, which includes job due dates, job processing times, job sequence-dependent setup time, and machine capacities. In section 3, we propose improving heuristics, which are the network algorithms merging with two different adjusting procedures respectively, for making the local optimum closer to global one. In section 4, we address a hybrid genetic algorithm in contrast to the two-phase heuristics we developed before. In the last section, we will describe the experimental framework and present the analysis of the results by comparing the genetic algorithm with two-phase heuristics.

## 2. Problem definition and formulation

Consider several product types of jobs with ready time and due date to be processed on identical parallel machines with capacity constraint. The job processing time depends on the product type of the job processed. Setup times for two consecutive jobs of different product type are sequence dependent. The objective is to find a schedule for the jobs which minimizes the maximum completion time without violating the due date restrictions and the machine capacity constraints.

We first define  $R = \{R_1, R_2, \dots, R_J, R_{J+1}\}$  as the  $J+1$  subsets (product types) of

jobs to be processed with each subset  $R_j = \{r_i | i = I_{j-1} + 1, I_{j-1} + 2, \dots, I_{j-1} + I_j\}$  containing  $I_j$  jobs, where  $I_0 = 0$  and  $I_{J+1} = K$ . Thus, job subset  $R_1 = \{r_1, r_2, \dots, r_{I_1}\}$  contains  $I_1$  jobs,  $R_2 = \{r_{I_1+1}, r_{I_1+2}, \dots, r_{I_1+I_2}\}$  contains  $I_2$  jobs, and  $R_{J+1} = \{r_{I_{J+1}}, r_{I_{J+2}}, \dots, r_{I_{J+K}}\}$  contains  $I_{J+1}$  ( $K$ ) jobs, where  $I_1 + I_2 + \dots + I_J = I$ . Let  $F$  be the total number of product families and  $J_f$  be the total number of product types in product family  $f$ , where  $\sum_{f=1}^F J_f = J$ . Then  $M = \{m_1, m_2, \dots, m_k\}$  can be defined as the set of machines containing  $K$  identical machines. The job subset  $R_{J+1}$ , which is a pseudo product type including  $K$  jobs, is used to indicate the  $K$  machines are in idle state. Therefore, there are  $I+K$  jobs grouped into  $J+1$  product types, at where the first  $I$  jobs are divided into  $J$  product types and the last  $K$  jobs are pseudo jobs. Let  $p_i$  be the processing time of job  $r_i$  in cluster  $R_j$  ( $r_i \in R_j$ ). Since the job processing time depends on the product type of the job, then  $p_i$  should be equal to  $p_{J(i)}$  given the function  $J(i)$  representing the product type of job  $r_i$ . Let  $Cmax$  be the maximum completion time (makespan) and  $W$  be the predetermined machine capacity expressed in terms of processing time units respectively. Let  $s_{i'}$  be the sequentially dependent setup time between any two consecutive jobs  $r_i$  and  $r_{i'}$ , in which  $s_{i'}$  is equal to  $s_{J(i)J(i')}$ .

Further, let  $x_{ik}$  be the variable indicating whether the job  $r_i$  is scheduled on machine  $m_k$ . If job  $r_i$  should be processed on machine  $m_k$ , set  $x_{ik} = 1$ , otherwise set  $x_{ik} = 0$ . Let  $t_{ik}$  be the starting time of job  $r_i$  to be processed on machine  $m_k$ . Set  $b_i$  as the ready time of job  $r_i$  and  $d_i$  as the due date of job  $r_i$ . The starting processing time  $t_{ik}$  should not be greater than the latest starting processing time  $e_i$ , which relates to the due date  $d_i$  and can be computed as  $e_i = d_i - p_i$ . The starting processing time  $t_{ik}$  also should not be less than the earliest starting processing time  $b_i$ . If job  $r_i$  is ready to be processed initially, then set  $b_i$  to 0. We note that the processing time and due dates for the job in  $R_{J+1}$  should be set to 0 so that these pseudo jobs can be scheduled as the first jobs on each machine, which indicate that each machine is initially in idle state. Let  $y_{i'k}$  be the precedence variable, where  $y_{i'k}$  should be set to 1 if the two jobs  $r_i$  and  $r_{i'}$  are scheduled on machine  $m_k$  and job  $r_i$  precedes job  $r_{i'}$  (not necessarily directly), and where  $y_{i'k} = 0$  otherwise. Further, let  $z_{i'k}$  be the direct-precedence variable,

where  $z_{ii'k}$  should be set to 1 if the two jobs  $r_i$  and  $r_{i'}$  are scheduled on machine  $m_k$  and job  $r_i$  precedes job  $r_{i'}$  directly, and where  $z_{ii'k} = 0$  otherwise.

To find a schedule for these jobs which minimizes makespan without violating the machine capacity and due date constraints, we consider the following integer programming model:

Minimize  $C \max$

subject to

$$\sum_{k=1}^K x_{ik} = 1, \text{ for all } i \quad (1)$$

$$\sum_{i=L+1}^{I+K} x_{ik} = 1, \text{ for all } k \quad (2)$$

$$\sum_{i=1}^{I+K} x_{ik} p_i + \sum_{i=1}^{I+K} \left( \sum_{i'=1}^{I+K} z_{ii'k} s_{i'} \right) \leq C \max, \text{ for all } k \quad (3)$$

$$\sum_{i=1}^{I+K} x_{ik} p_i + \sum_{i=1}^{I+K} \left( \sum_{i'=1}^{I+K} z_{ii'k} s_{i'} \right) \leq W, \text{ for all } k \quad (4)$$

$$t_{ik} + p_i + s_{ii'} - t_{i'k} + Q(y_{ii'k} - 1) \leq 0, \text{ for all } i, k \quad (5)$$

$$t_{ik} + p_i + s_{ii'} - t_{i'k} + Q(y_{ii'k} + z_{ii'k} - 2) \geq 0, \text{ for all } i, k \quad (6)$$

$$t_{ik} \geq b_i x_{ik}, \text{ for all } i, k \quad (7)$$

$$t_{ik} \leq e_i x_{ik}, \text{ for all } i, k \quad (8)$$

$$(y_{ii'k} + y_{i'ik}) - Q(x_{ik} + x_{i'k} - 2) \geq 1, \text{ for all } i, k \quad (9)$$

$$(y_{ii'k} + y_{i'ik}) + Q(x_{ik} + x_{i'k} - 2) \leq 1, \text{ for all } i, k \quad (10)$$

$$(y_{ii'k} + y_{i'ik}) - Q(x_{ik} + x_{i'k}) \leq 0, \text{ for all } i, k \quad (11)$$

$$(y_{ii'k} + y_{i'ik}) - Q(x_{i'k} - x_{ik} + 1) \leq 0, \text{ for all } i, k \quad (12)$$

$$(y_{ii'k} + y_{i'ik}) - Q(x_{ik} - x_{i'k} + 1) \leq 0, \text{ for all } i, k \quad (13)$$

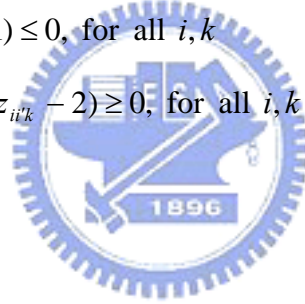
$$y_{ii'k} \geq z_{ii'k} \text{ for all } i, k \quad (14)$$

$$\sum_{i=1}^{I+K} x_{ik} - \sum_{i \neq i'} z_{ii'k} = 1, \text{ for all } k \quad (15)$$

$$y_{ii^*k} + z_{ii^*k} - Q(y_{ii^*k} + z_{ii^*k} - 2) - Q(y_{ii'k} - z_{ii'k} - 1) \geq 2, \text{ for all } i, k \quad (16)$$

$$x_{ik} \in \{0,1\}, \text{ for all } i, k \quad (17)$$

$$y_{ii'k} \in \{0,1\}, \text{ for all } i, k \quad (18)$$



$$z_{ii'k} \in \{0,1\}, \text{ for all } i,k \quad (19)$$

The constraints in (1) guarantee that job  $r_i$  is processed by one machine exactly once. The constraints in (2) guarantee that only one pseudo job  $r_i$ ,  $I+1 \leq i \leq I+K$ , is scheduled on a machine. The constraints in (3) state that each machine workload does not exceed the maximum completion time  $C_{max}$  among all  $K$  machines. The constraints in (4) state that each machine workload does not exceed the machine capacity  $W$ . The constraints in (5) and (6) ensure that  $t_{ik} + p_i + s_{ii'} = t_{i'k}$  if job  $r_i$  precedes job  $r_{i'}$  directly ( $y_{ii'k} = 1$  and  $z_{ii'k} = 1$ ). The constraints in (5) ensure the satisfaction of the inequality  $t_{ik} + p_i + s_{ii'} \leq t_{i'k}$  if job  $r_i$  preceding job  $r_{i'}$  ( $y_{ii'k} = 1$ ). The number  $Q$  is a constant, which is chosen to be sufficiently large so that the constraints in (5) are satisfied for  $y_{ii'k} = 0$  or  $1$ . For example, we can choose  $Q = \sum_{i=1}^I (p_i + \max_{i'} \{s_{ii'}\})$ . The constraints in (6) ensure the satisfaction of the inequality  $t_{ik} + p_i + s_{ii'} \geq t_{i'k}$  and the event the jobs  $r_i$  proceeding job  $r_{i'}$  directly ( $y_{ii'k} + z_{ii'k} - 2 = 0$ ).

The constraints in (7) and (8) state that the starting processing time  $t_{ik}$  for each job  $r_i$  scheduled on machine  $m_k$  ( $x_{ik} = 1$ ) should not be less than the earliest starting processing time  $b_i$  and not be greater than the latest starting processing time  $e_i$ . The constraints in (9) and (10) ensure that one job should precede another ( $y_{ii'k} + y_{i'ik} = 1$ ) if two jobs are scheduled on the same machine ( $x_{ik} + x_{i'k} - 2 = 0$ ). The number  $Q$  is a constant, which is chosen to be sufficiently large so that the constraints in (9) and (10) are satisfied for  $x_{ik} + x_{i'k} - 2 < 0$ . The constraints in (11) ensure that the precedence variables  $y_{ii'k}$  and  $y_{i'ik}$  should be set to zero ( $y_{ii'k} + y_{i'ik} \leq 0$ ) if any two jobs  $r_i$  and  $r_{i'}$  are not scheduled on the machine  $m_k$  ( $x_{ik} + x_{i'k} = 0$ ). The constraints in (12) and (13) ensure that the precedence variables  $y_{ii'k}$  and  $y_{i'ik}$  should be set to zero ( $y_{ii'k} + y_{i'ik} \leq 0$ ) if any one job  $r_i$  or  $r_{i'}$  is not scheduled on the machine  $m_k$ . The constraints in (12) indicates the case that job  $r_i$  is scheduled on machine  $m_k$  and the job  $r_{i'}$  is scheduled on another machine ( $x_{i'k} - x_{ik} + 1 = 0$ ) and the constraints in (13) indicates the case that job  $r_{i'}$  is scheduled on machine  $m_k$  and the job  $r_i$  is scheduled on another machine ( $x_{ik} - x_{i'k} + 1 = 0$ ).

The constraints in (14) ensure that job  $r_i$  could precede job  $r_{i'}$  directly ( $z_{ii'k} = 1$ )

only when  $y_{i'k} = 1$  and job  $r_i$  could not precede job  $r_{i'}$  directly ( $z_{i'k} = 0$ ) if job  $r_i$  is scheduled after job  $r_{i'}$  ( $y_{i'k} = 0$ ). The constraints in (15) state that there should exist I-1 directly precedence variables, which are set to 1 on the schedule with I jobs. The constraints in (16) state that when the job  $r_i$  precedes job  $r_{i'}$  but not consecutively ( $y_{i'k} = 1$  and  $z_{i'k} = 0$ ), then there must exist another job  $r_{i^*}$  scheduling after job  $r_i$  directly ( $y_{i^*k} = 1$  and  $z_{i^*k} = 1$ ) and ensuring the satisfaction of the inequality  $y_{i^*k} + z_{i^*k} \geq 2$ .

### 3. The Improving Heuristics

There are many kinds of network algorithms for WSPS with minimizing total machine workload. VRPTW algorithms are one of them which have been successfully applied to solve WSPS with good efficiency. Because VRPTW algorithms are effective for solving WSPS, we adopt these WSPS algorithms to solve WSPS with minimum makespan in the following of this paper. We use these algorithms based on expected machine load  $EL$  restriction to find feasible solutions of WSPS with minimum makespan in phase I of improving heuristics. Then feasible solutions would be improved through the adjusting procedure, phase II of improving heuristics. In this paper we propose two-phase heuristics, improving heuristics, to help solving the WSPS with minimum makespan more efficiently. The main idea is that improving heuristics use the adjusting procedure to improve feasible solutions solved by WSPS algorithms. The improving procedure would search the best solution through the different machine workload repeatedly. Phase one of the improving heuristic is to apply some efficient WSPS algorithms based on the expected machine load  $EL$  restriction for finding feasible solutions of WSPS with minimum makespan. In phase two, we will provide an improving procedure for making the local optimum solved by WSPS algorithms climbing to the global one. The structure of the improving heuristic is shown as Fig. 1.

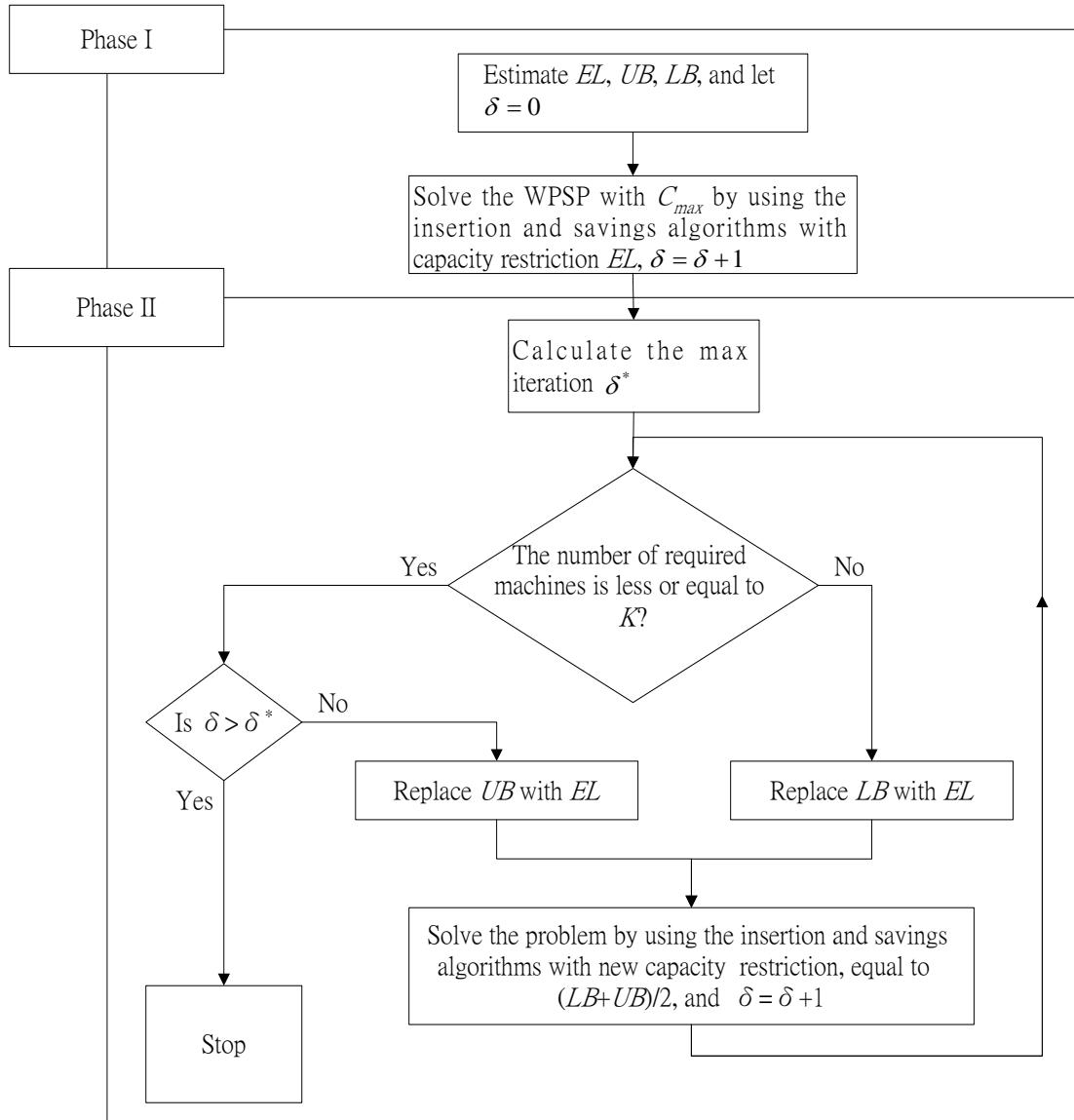


Figure 1. The structure of the improving heuristic

To find the minimal  $C_{max}$  more efficiently, we provide the estimation of the expected machine workload  $EL$  corresponding to the parallel machine scheduling problem, which will be utilized along with scheduling algorithms. Before calculating  $EL$ , we would define the notation  $ES$ , which is the total expected setup time of all identical parallel machines. The expected machine workload  $EL$  equals to the sum of the expected setup time  $ES$  and the total job processing time divided by the number of available machines  $K$ . Let  $s_{i'}$  be the sequence dependent setup time of any two consecutive jobs  $r_i$  and  $r_{i'}$  on the same machine. Let  $Max[s_{U_i}]$  be the maximal setup time of machine switching from idle status (denoted by the label “U”) to processing status,  $Max[s_{i,U}]$  be the maximal setup time of machine switching from



processing status to idle status,  $Max1[s_{i'j}]$  be the maximal setup time of two consecutive jobs processed on machine coming from different product type and different product family,  $Max2[s_{i'j}]$  be the maximal setup time of two consecutive jobs from different product type and same product family. Therefore, equation (20) expresses the computation of  $ES$ .

$$\begin{aligned}
 ES = & K \times (Max[s_{U_i}] + Max[s_{iU}]) + \left(\frac{I}{K} - 1\right) \times (Max1[s_{i'j}] \times \frac{\sum_{f=1}^F C_1^{J_f} \times (J - C_1^{J_f})}{P_2^J + C_1^J}) \\
 & + Max2[s_{i'j}] \times \frac{\sum_{f=1}^F P_2^{J_f}}{P_2^J + C_1^J} + 0 \times \frac{\sum_{f=1}^F C_1^{J_f}}{P_2^J + C_1^J}
 \end{aligned} \quad (20)$$

where the notations ‘‘P’’ and ‘‘C’’ in equation (20) represent the symbol of permutation in statistics. And the coefficient ‘‘0’’ indicates the setup time of two consecutive jobs from the same product type and the same product family should be zero.

Let the parameter  $\sigma$ , which may vary according to the problem data structure, be the allowance of uniform capacity decided by the user and be set as the value between -0.5 and 0.5. We consider the expected capacity  $EL$  is the allowance multiplied by the outcome, which is the sum of total job processing times plus the expected setup time  $ES$  divided by  $K$  machines fairly. Then we can get the  $EL$  as follows:

$$EL = \left[ \frac{1}{K} \times \left( \sum_{i=1}^I p_i + ES \right) \right] \times (1 + \sigma) \quad \text{and} \quad -0.5 \leq \sigma \leq 0.5 \quad (21)$$

### 3.1 Phase I - Existing network algorithms

Generally speaking, the WSPSP algorithms include insertion and saving algorithms. The insertion algorithms generally include two types, the sequential and the parallel. The saving based procedures include four types, the sequential, the parallel, the generalized, and the matching based. We first define the order  $(i_{0k}, i_{1k}, \dots, i_{(\nu-1)k}, i_{\nu k}, \dots, i_{Uk})$  where  $i_{0k}$  and  $i_{Uk}$  both represent the machine  $m_k$  is in the idle state, and  $i_{\nu k}$  represents that selected job has been scheduled on the  $\nu$ th position of machine  $m_k$ . Then we would review savings and insertion algorithms

by citing Clark and Wright [17], Golden [18], Pearn et al. [19], Solomon [20], Potvin and Rousseau [21], and Yang et al. [22]. These algorithms are including the sequential saving algorithm, the modified sequential saving algorithm, the sequential insertion algorithm, the parallel insertion algorithm, and modified parallel insertion algorithms. The procedures of these algorithms above are introduced as follows.

### Sequential savings algorithm (SSA)

First of all, the sequential savings algorithm calculates the savings of all paired-jobs and creates a saving list by arranging their saving values in descending order. Then we pick the first pair of jobs from the top of the saving list to start an initial schedule. We can confirm whether a selected pair of jobs is feasible by checking the machine capacity constraint and the due date restrictions of jobs. The sequential savings algorithm spreads out the schedule by finding the feasible pair of jobs from the top of the savings list and adding it to either one endpoint of the schedule. If the current schedule is too tight to add any job in, choose the feasible pair of jobs from the top of the saving list as a new schedule. Repeat this step until all jobs are scheduled. The procedure is presented in the following.

Step 1. (Initialization) Calculate the savings value  $SA_{ii'}$  defined as the following for all pairs of jobs  $r_i$  and  $r_{i'}$ , where U denotes the idle status.

$$SA_{ii'} = s_{Ui} + s_{iU} + s_{Ui'} + s_{i'U} - (s_{Ui} + s_{ii'} + s_{i'U}) = s_{iU} + s_{Ui'} - s_{ii'} \quad (22)$$

Step 2. Arrange the savings and create a list of the saving values in a descending order.

Step 3. Choose the first pair of jobs from the saving list as an initial schedule. Start from the top of the savings list, and proceed with the following sub-steps:

Step 3-1. Select the first pair from the top saving list without violating the machine capacity and due date constraints. Then add it to either one end of the current schedule.

Step 3-2. If the current schedule is too tight to add any job on it, choose the best feasible pair on the saving list to start a new schedule.

Step 4. The chosen jobs then form a feasible machine schedule. Repeat step 3 until all the jobs on the saving list are scheduled.

### Modified sequential savings algorithm (MSSA)

The modified sequential savings algorithm adds two terms into the savings estimation, the consideration of the postponement and the time window restriction. For the postponement, the selecting way would tend to choose the pair of jobs with not only higher saving values but longer processing time. By this way, the jobs with longer processing time are forced to be processed earlier than others with shorter one. Considering the other term, the job with earlier job latest starting time  $e_i$  would be placed before the job with later one  $e_j$  on the savings list. Two parameters,  $A$  and  $B$ , are added into the savings function to present the percentage of postponement and time window restriction, and  $W$  is the predetermined capacity. The new savings function is expressed in the following:

$$MSA_{ii'} = A(s_{iU} + s_{U'i'} - s_{ii'}) + (1-A)p_i + W\left(\frac{B}{e_i} - \frac{(1-B)}{e_i'}\right), \quad 0 < A < 1, 0 < B < 1 \quad (23)$$

### Sequential insertion algorithm (SIA)

The main part of the sequential insertion is to build one schedule once until all jobs are scheduled. The sequential insertion would find the maximal benefit among the schedule places that a selected job can insert into. When the existing schedule is full of jobs, we create another new machine schedule. The initial rule is to select a job with the maximal initial setup time. After initializing the current schedule, the priority of selecting job depends on the regret value  $c_2(u)$  of all unscheduled jobs. Find the best insertion place  $c'_1(u, k, v^*)$  of all unscheduled jobs and select job  $u^*$  with the largest regret value  $c'_2(u^*)$  as the first inserted job. The evaluations of insertion cost and regret values are defined as follows.

$$c_1(u, k, v) = c_1(i_{(v-1)k}, u, i_{vk}) = s_{i_{(v-1)k}u} + s_{ui_{vk}} - s_{i_{(v-1)k}i_{vk}} \quad (24)$$

$$c'_1(u, k, v^*) = \min_v [c_1(u, k, v)] \quad (25)$$

$$c_2(u) = s_{Uu} - c'_1(u, k, v^*) \quad (26)$$

$$c'_2(u^*) = \max_u [c_2(u)] \quad (27)$$

In equation (24), the jobs  $i_{(v-1)k}$  and  $i_{vk}$  are placed individually on the  $(v-1)$ th and  $v$ th positions of machine  $m_k$ . And the term  $s_{Uu}$  of equation (26) represents the initial setup time of the unscheduled job  $r_u$ . For each unscheduled job, we first compute its best feasible insertion place  $c'_1(u, k, v^*)$  in equation (25), and we can get the regret value of  $c_2(u)$  in equation (26). The job with larger value  $c_2(u)$  should have the priority to be scheduled. Therefore, select the job  $u^*$  with largest value  $c'_2(u^*)$  and insert it into the best position of the schedule. All unscheduled jobs will be inserted under the following procedure.

- Step 1. Initialize the schedule by selecting the job with the maximal initial setup time.
- Step 2. For each unscheduled job, compute the best feasible insertion place  $v^*$ , which has the smallest insertion value  $c'_1(u, k, v^*)$  on machine  $m_k$ .
- Step 3. Select the unscheduled job  $u^*$  with the largest value  $c_2(u^*)$  and put it into the best insertion place of the schedule. If the existing schedule is too full to add any unscheduled job, create a new schedule on another machine.
- Step 4. Repeat Step 2 and Step 3 until all jobs are scheduled.

Parallel insertion algorithm (PIA)

The parallel insertion algorithm constructs a set of initial schedules on all machines in the beginning. Besides, it also creates a new regret measure, which is the sum of absolute differences between the best alternative on one machine and other alternatives on other machines. A large regret measure means that there is a large gap between the best insertion place of the unscheduled job on one machine and its best insertion place on the other machines. Hence, unscheduled jobs with larger regret values should be inserted into the schedule first, because there are large cost differences of the best insertion place and second alternative. In this algorithm, we add two criteria into our selecting rule. One is the value  $c''_1(u, k^*, v^*)$ , which has the smallest insertion cost on  $v^*$ th position of machine  $k^*$ . The other is the regret value  $c_3(u)$  different from  $c_2(u)$  of equation (26). The insertion functions are as follows.

$$c_1''(u, k^*, v^*) = \min_k [c_1'(u, k, v^*)] \quad (28)$$

$$c_3(u) = \sum_{k \neq k^*} [c_1'(u, k, v^*) - c_1''(u, k^*, v^*)] \quad (29)$$

$$c_3'(u^*) = \max_u [c_3(u)] \quad (30)$$

Initialization is done by selecting the unscheduled jobs with the first  $K$  largest initial setup times and putting them into the initial schedule of each machine. By applying this method, we can get the  $K$  initial schedules and compute the best insertion place in each of the schedules for all unscheduled jobs. Then we compute the regret value  $c_3(u)$  of all unscheduled jobs and find the largest value  $c_3'(u^*)$  of job  $u^*$ . Select job  $u^*$  and insert it into the best position  $v^*$  of machine  $k^*$  with value  $c_1''(u, k^*, v^*)$ . The procedure of PIA is described as below.

- Step 1. Initialize the schedule on each machine by selecting  $K$  jobs with the first  $K$  largest initial setup times.
- Step 2. For each unscheduled job, find its best feasible insertion place by computing  $c_1''(u, k^*, v^*)$ .
- Step 3. Calculate the regret value  $c_3(u)$  for each unscheduled job. Select the job  $u^*$  with the largest regret measure  $c_3'(u^*)$  among all unscheduled jobs. Insert it into the  $v^*$ th position of machine  $k^*$  without violating the machine capacity and its due date restrictions.
- Step 4. Repeat Step 2 and Step 3 until all jobs are scheduled.

#### Parallel insertion with new initial criteria (PIA I)

According to idea of the VRPTW, PIA first selects the farthest node to visit at the beginning stage. However, selecting the job with largest initial setup time, the farthest node, may not reduce the total machine workload. Comparing to PIA, this modified one adds new initial criteria in find initial jobs of parallel machines. Inserting a job into the existing schedule of the same product family can significantly reduce the increased setup time. Because the jobs of the same product type must belong to the same family, this procedure chooses the product type including the maximal number of jobs and picks the job with the smallest latest starting time  $e_i$  of this type to be the initial schedule on each machine. Once the job  $r_i$  of product

type  $J(i)$  is selected for a specific machine, other jobs of product type  $J(i)$  cannot be selected as the initial schedules on other machines. After initializing  $K$  schedules of machines, the following steps of PIA I are identical to PIA.

#### Parallel insertion with the slackness (PIA II)

In order to express the impact of job due date, this algorithm adopts the modified insertion functions  $c_{11}(u, k, \nu)$  of equation (31) instead of the value  $c_1(u, k, \nu)$  of PIA. It adds the latest starting time  $e_u$  of unscheduled job  $u$  into consideration and that would make the selection rule choose the job with smaller latest starting time as the priority possibly. The modified insertion function is as the following.

$$c_{11}(u, k, \nu) = \lambda(s_{i_{(v-1)k}u} + s_{ui_k} - s_{i_{(v-1)k}i_k}) + (1 - \lambda)(e_u), \quad 0 \leq \lambda \leq 1 \quad (31)$$

According to the insertion function above, we can determine the ratio of the insertion values that the slackness would have by revising the value  $\lambda$ . The insertion procedure is the same as PIA.

#### Parallel insertion with new initial criteria and slackness (PIA III)

Because PIA I and PIA II do have the advantage of reducing total machine setup time, we generate the new modified parallel insertion algorithm by combining two insertion criteria of job selection. At the beginning of inserting initial jobs, the algorithm selects the product type including the maximal number of jobs and chooses the job with the smallest value  $e_i$  in this product type as the initial schedule on each machine. Once the job  $r_i$  of product type  $J(i)$  is selected for a specific machine, other jobs of product type  $J(i)$  cannot be put as the initial schedule on other machines. The insertion cost is the same as  $c_{11}(u, k, \nu)$  in equation (31) and other steps in this modification algorithm are identical to PIA II.

#### Parallel insertion with the variance of regret measure (PIA IV)

The original parallel insertion procedure does not consider the impact of the

variance between the best insertion places on all machines. This modified algorithm creates a new regret measure including not only the absolute total differences of the best insertion value and other alternatives but also the variance among them. Therefore, the selected job would have a significant variance  $Var(c_1)$  under large regret values. The modified calculation of the regret value is the following.

$$c_4(u) = \varphi \sum_{k \neq k^*} [c'_1(u, k, v^*) - c''_1(u, k^*, v^*)] + (1 - \varphi)Var(c_1), \quad 0 \leq \varphi \leq 1 \quad (32)$$

$$Var(c'_1) = \left[ \sum_{k=1}^K (c'_1(u, k, v^*) - Avg(c'_1))^2 \right] (K - 1)^{-1} \quad (33)$$

$$Avg(c'_1) = \left[ \sum_{k=1}^K c'_1(u, k, v^*) \right] K^{-1} \quad (34)$$

The notation  $Var(c'_1)$  in equation (33) is the variance of best insertion cost between all parallel machines, and  $Avg(c'_1)$  is the average value of best insertion cost on all parallel machines. We can determine the schedule ranking of all jobs on all parallel machines by adjusting the parameter  $\varphi$ .

### 3.2 Phase II - Network Adjusting Procedure

In accordance with WPSP with minimum makespan, we develop an adjusting procedure for WPSP algorithms. The adjusting procedure is proceeding based on the concept of lower bound  $LB$  and upper bound  $UB$  constraints. The adjusting procedure is searching a near-optimal solution under the lower bound and upper bound, so the solution time of the adjusting procedure is longer than Phase I. Here are the steps of the adjusting procedure we developed.

#### Adjusting Procedure

The procedure makes the basic solutions solved by phase I closer to the optimum based on lower bound and upper bound restriction. Let  $ML_k$  be the total machine load on machine  $m_k$ , and let  $\delta$  be the number of adjusting expected machine load  $EL$ . Here comes the detail of the adjusting procedure.

Step 1. Let  $\delta$  be zero and set the value  $EL^0$  be the same of  $EL$ . Use the WPSP algorithm described to generate a basic solution by adding a restriction of

capacity  $EL^0$  into it.

Step 2. Here we define the initial lower bound and upper bound as follows.

$$LB^0 = \min_{k \in K} \{ML_k\} \quad (35)$$

$$UB^0 = \max_{k \in K} \{ML_k\} + \max_{i \in I} \{p_i\} \quad (36)$$

According to the property of the solution, we define the maximal number  $\delta^*$  of adjusting in two ways.

(1) If the actually required number of machines  $k$  is smaller than the number of available machines  $K$ , we let  $\delta^* = \log_2(EL^0 - LB^0)$ , which represents the maximal adjusting times we can use in this adjusting procedure.

(2) If the actually required number of machines  $k$  is larger than the number of available machines  $K$ , we let  $\delta^* = \log_2(UB^0 - EL^0)$  in another way.

Step 3. The step is divided into two different decision rules.

(1) If the actually required number of machines  $k$  is smaller than the number of available machines  $K$ , set the value  $UB^\delta$  to be the same of  $EL^\delta$ , and update  $\delta$  with the increment 1. The expected machine load is replaced with  $(LB^{\delta'} + UB^{\delta'})/2$ , where  $\delta' = \delta - 1$ , that is,  $EL^\delta = (LB^{\delta'} + UB^{\delta'})/2$ .

(2) If the actually required number of machines  $k$  is larger than the number of available machines  $K$ , set the value  $LB^\delta$  to be the same of  $EL^\delta$ , and update  $\delta$  with the increment 1. The expected machine load is replaced with  $(LB^{\delta'} + UB^{\delta'})/2$ , where  $\delta' = \delta - 1$ . The adjusting equation is the same as the one above.

Step 4. Restart the network algorithm with the restriction of adjusting value  $EL^\delta$ .

Step 5. If the value of  $\delta$  is larger than  $\delta^*$  and the  $\delta$ th scheduling solution is feasible, stop the procedure. Otherwise ( $\delta$  is less than  $\delta^*$  or the  $\delta$ th scheduling solution is infeasible) go to step 3 until all jobs are scheduled.

#### 4. The Genetic Algorithm for WPSP with $\min C_{max}$

A genetic algorithm is a search algorithm based on the mechanism of genetics and evolution, which combines the exploitation of past results with the information of new areas of the search space. A genetic algorithm can imitate some innovative talent of a human search by using the surviving techniques of fitness function. The mechanism of a genetic algorithm is very simple, involving nothing but copying strings and swapping positions among strings. In every new generation of a genetic algorithm, a set of strings are created exploiting information from the previous ones. With this collection of artificial strings, a new part of population is tried for good measure and the best overall solution would become the candidate solution to the



problem.

#### **4.1 GA for Parallel Machine Problem**

The genetic algorithm takes advantage of historical information effectively to proceed with new search points for expected improvement. Simple operation and effective power are two primary attractions of the GA approach. The effectiveness of GA depends on an appropriate mix of exploration and exploitation. Two genetic operators, crossover, and mutation, are designed to approach this goal. Many researchers have considered the parallel-machine scheduling problem by using the genetic approach. Zomaya and Teh [23] employed a GA considering load balancing issues suchlike threshold policies, information exchange criteria, and inter-processor communication, to solve the dynamic load balancing problem with minimizing the maximum completion time. Cheng et al. [24,27] considered an identical parallel machine system with an objective of minimizing the maximal weighted absolute lateness and proposed a hybrid algorithm which combined the GA with the due date determination. They proved that mutation should play more critical role than the crossover and the hybrid genetic algorithm did outperform the conventional heuristics. Min and Cheng [25] provided a genetic algorithm based on the machine code for minimizing makespan in identical parallel machine scheduling problem and it was fit for larger scale problems with comparison to LPT and SA. Cochran et al. [26] proposed a two-stage multi-population genetic algorithm (MPGA) to solve parallel machine with multiple objectives. Multiple objectives are combined via the multiplication of the relative measure of each objective in the first stage, and the solutions of the first stage are arranged into sub-population to evolve separately under the elitist strategy. Ulusoy et al. [28] proposed a genetic algorithm with the crossover operator MCUOX for solving the parallel-machine scheduling problem with minimizing the total weighted earliness and tardiness values. They showed that GA with MCUOX outperformed in larger-sized, more difficult problems. Herrmann [29] provided a two-space genetic algorithm representing solutions and scenarios for solving minimizing makespan problems and the experiment showed the two-space GA could find robust solutions. Tamaki et al. [30] dealt with identical parallel

machine scheduling problems with the objective of minimizing total flow time and earliness/tardiness penalties. They proposed a genetic algorithm combined with a simplex method to generate an effective set of Pareto-optimal schedules. Vignier et al. [31] provided a hybrid method to solve a parallel-machine scheduling problem with minimizing the total cost of assignment and setup time and the result showed efficient in industrial case. Lin [32] considered a unrelated parallel machine scheduling problem with due date restriction for minimizing makespan, total weighted tardiness, and total weighted flow time. She proposed a genetic algorithm combined with prescribed initialization for solving the multi-objective and the result expressed that the GA with prescribed initialization could find an optimal solution in small sized problems. Table 1 shows the differences of GA these researchers developed.



Table 1. The comparison of GA under various problem characters, crossover, and mutation.

Author	Objective function of problem	Due date consideration	Setup time consideration	Release time consideration	Coding method	Crossover operator	Mutation operator	Initial population generation	GA compared with:	GA results
Zomaya, A.Y. and Teh, Y.H.	Minimize makespan and processor utilization	No	No	Yes	Each element in the string has two decimal values, one to represent the task number and the other to represent the size of the task	The cycle crossover method	Swap mutation is adopted by randomly selecting two tasks and then swapping them	Use a dynamic load-balancing algorithm	GA itself for different window size, population size, and generation size	GA performs better under window size=20, generation number=20, and population size=30
Cheng, R., Gen, M., and Tozawa, T.	Minmax the weighted absolute lateness	A common due date	No	No	Use a list of job symbol and partitioning symbol as the coding scheme	The sub-schedule preservation crossover	Use random exchange by selecting two random genes and then exchanging their positions	Random generate	Existing heuristic procedure	GA has a better result with considering mutation
Min, L. and Cheng, W.	Minimize makespan	No	No	No	The gene code is $k_1, k_2, \dots, k_j, \dots, k_n$ , where $k_j \in [1, m]$ . It's the number of the machine on which each job is processed	The two-point crossover	A digit is selected according to pre-defined mutation probability and replaced with a different number	Random generate	LPT and SA	GA is more efficient and fit for larger scale problem
Cochran, J.K., Horng, S.M., and Fowler, J.W.	Minimize makespan, total weighted tardiness, and total weighted completion time	Yes (each job has its own due date)	No	Yes (each job has its own release time)	The number in position one of the chromosome represents the machine that will process job one	The one-point crossover	A digit is selected according to pre-defined mutation probability and replaced with a different number	Random generate	Multi-objective genetic algorithm (MOGA)	Multi-population genetic algorithm (MPGA) shows better results over a wide range of problems

Table 1. Continued

Author	Objective function of problem	Due date consideration	Setup time consideration	Release time consideration	Coding method	Crossover operator	Mutation operator	Initial population generation	GA compared with:	GA results
Cheng, R. and Gen, M.	Minimize the maximum weighted absolute lateness	A common due date	No	No	Use a list of job symbol and partitioning symbol as the coding scheme	The sub-schedule preservation crossover	Use random exchange by selecting two random genes and then exchanging their positions	Random generate	The published GA and the heuristic, V-shape policy	The memetic algorithm outperforms both GA and the heuristic
Serifoglu, F.S. and Ulusoy, G.	Minimize total weighted earliness and tardiness	Yes (each job has its own due date)	Yes (sequence dependent)	Yes (each job has its own release time)	The chromosome representation incorporates both the sequencing and the machine selection	The multi-component uniform order-based crossover	Swap mutation randomly chooses two positions on the chromosome and swap the contents. Bit mutation is applied to each gene in the population with a very low probability	Three scheduling rules (forward-pass, backward-forward-pass, and 3-D scheduling rules)	Four different scheduling rules and two distinct crossovers	GA with GA-NCUOX performs well in larger-sized problems
Herrmann, J.W.	Minimize makespan	No	No	No	The GA uses 27-bit strings and each individual in the population has nine genes, one for each job	—	—	Use two-space GA to generate	Worst case optimization	Two-space GA is able to find robust solutions
Tamaki, H., Nishino, E., and Abe, S.	Minimize total flow time, total weighted earliness and tardiness	Yes (each job has its own due date)	No	Yes (each job has its own release time)	An individual is represented as a combination of two sub-string with the length of the number of jobs	The one-point crossover	One locus is selected randomly, and then the gene is changed to another gene with a prescribed probability	Random generate	Different setting of the weight parameters	The effectiveness for generating a variety of Pareto-optimal schedules is investigated

Table 1. Continued

Author	Objective function of problem	Due date consideration	Setup time consideration	Release time consideration	Coding method	Crossover operator	Mutation operator	Initial population generation	GA compared with:	GA results
Vignier, A., Sonntag, B., and Portmann, M-C.	Minimize the total costs of assignment and setup time	Yes (each job has its own due date)	Yes (sequence dependent)	Yes (each job has its own release time)	The genome is composed of two chromosome, one for the assignment and the other for the sequence	The one-point crossover and the edge recombination operator	Swap two consecutive jobs or change the assignment of a job in an individual	Use two presented heuristics	Two presented Heuristics	The results of GA are better than those they have obtained before
Lin, C.C.	Minimize makespan, total weighted flow time, and total weighted tardiness	Yes (each job has its own due date)	No	No	Use a list of job symbol and partitioning symbol as the coding scheme	The sub-schedule preservation crossover	Use random exchange by selecting two random genes and then exchanging their positions	Use EDD/WSPT(mean), WSPT(mean)/EDD, SPT(mean)/EDD, LPT(mean)/EDD, MDD, SPT(min)/EDD, LPT(max)/EDD, and some heuristics to generate	Branch & bound and traditional GA	The modified GA can find the optimal solution in small-sized problems, and have a better solution speed than branch & bound

## 4.2 The Hybrid Genetic Algorithm

GA is an artificial adaptive system for simulating natural evolution. Because of their effectiveness and efficiency in searching complex spaces, they are increasingly used to attack NP-hard problems. The core of GA is its crossover operator that progressively constructs near optimal solutions from good feasible solutions. In this paper, we propose a new crossover that protects better schedules of machines from elimination. First of all, we would give some definition of terms including  $Pop\_size$ ,  $Ex\_Pop\_size$ ,  $max\_gene$ ,  $off\_size$ , and  $p_m$ . Let notations  $Pop\_size$  and  $Ex\_Pop\_size$  be the number of parents and extended population individually,  $off\_size$  be the number of offspring, and  $max\_gene$  be the maximum number of generated generation. Besides, let  $p_m$  be the prescribed probability of mutation. The flowchart of the execution for hybrid genetic algorithm is given in Fig. 2.



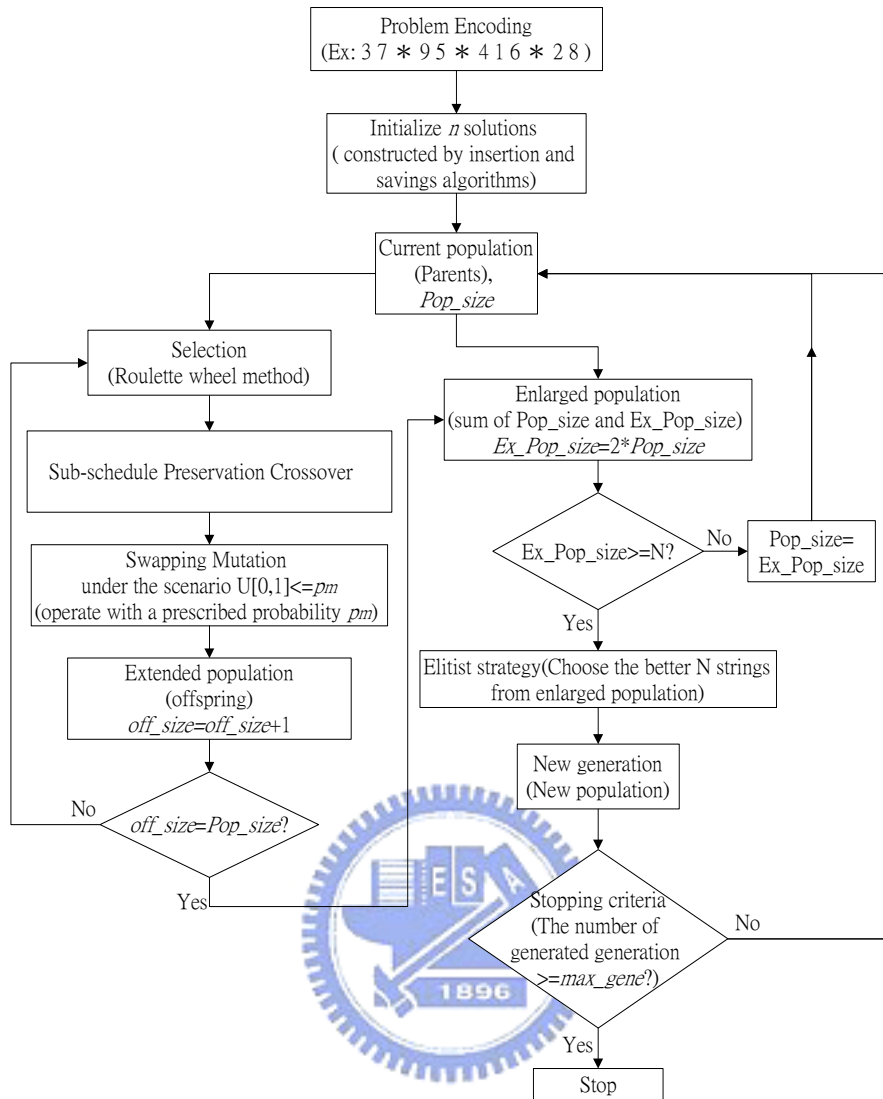


Figure 2. The flowchart of the execution for hybrid genetic algorithm.

#### 4.2.1 Problem encoding

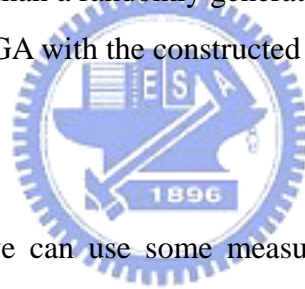
Normal binary encoding does not work very well for the parallel-machine scheduling problem because the encoding strings may become too redundant to incorporate all needed messages. Therefore, we code the strings by using the representation of decimal numbers. We use a set of integers and stars \* representing the job identity and the partition of jobs to machines for parallel-machine scheduling problem. The integers and stars on the string represent all possible sequences of jobs on parallel machines. For a problem of  $n$  jobs and  $m$  parallel machines, a correct chromosome must consist of  $n$  job symbols and  $m-1$  partitioning symbols \*, which mean there should be  $n+m-1$  genes in a chromosome. We can

give a simple example of 9 jobs and 4 parallel machines shown below. The string can be represented as follows:

[ 3 7 \* 9 5 \* 4 1 6 \* 2 8 ]

#### 4.2.2 Initialization

In each generation the GA manipulates a set of operators in the population. The construction of the initial population is important since the operators of GA would preserve some part of better chromosomes generation to generation. The initial population influences not only the convergence of the GA but the qualities of chromosomes generated. The initial chromosomes are constructed with network algorithms we described in this identical parallel-machine scheduling problem. We are looking for the better near optimal solution produced by the initial population based on network algorithms than a randomly generated population. We also predict the quick convergence of the GA with the constructed initial population.



#### 4.2.3 Selection

During each generation, we can use some measure of functions to evaluate the values of chromosomes. Fitness is estimated based on the objective function in most cases of optimization problems. As the objective of our problem is WSPS with minimizing makespan, we can use the reciprocal of the objective function as the fitness value. So a fitter chromosome has a larger fitness value. The fitness value of each chromosome is defined as following:

$$F(\alpha, \beta) = (Cmax_{\alpha, \beta})^{-1} \times \left[ \frac{K - k + 1}{K \times Q} \right] \quad (37)$$

where the term  $Cmax_{\alpha, \beta}$  expresses the makespan of the  $\alpha$  th chromosome in the pool when the GA cycle proceeds to the  $\beta$  th generation. The term  $K$  represents the available machine number and  $k$  represents the actual required machine number. Besides, the term  $Q$  is a constant described in section 2 for keeping the calculation  $(K - k - 1) / KQ$  moving around 0 and 1. So the function  $\lceil K - k / KQ \rceil$  can make sure that the fitness value calculated by equation (37) is available to be used.



The selection technique in this paper is based on the roulette wheel method. In this case, the probabilities of the individual chromosomes surviving to the next generation determine the slots of the roulette wheel. These probabilities of these slots on the roulette wheel are estimated by dividing the fitness value of each chromosome by the sum of the fitness values of all chromosomes in the current population. Cumulating the probabilities of each chromosome creates the individual slots. Here comes an example, which calculates the individual slots on the roulette wheel. There are three chromosomes in the population, of which the probabilities of chromosomes are 0.2, 0.3, and 0.5 individually. Then the slots of chromosome 1, chromosome 2 and chromosome 3 will range from 0-0.2, 0.2-0.5, and 0.5-1 respectively. Each slot size of chromosome will be proportional to its fitness value.

#### **4.2.4 Genetic operators**

Two genetic operators, crossover and mutation, are usually used in the genetic algorithm. Crossover generates offspring by combining two chromosomes' features. Mutation operates one chromosome by randomly selecting two genes and swapping them. Generally speaking, the crossover operator plays an important role for the performance in the GA cycle. The performance of crossover in each operator does affect the performance of GA. So we adopt the different rules for designing crossover and mutation. Both crossover and mutation can handle the job permutation and setup time on the identical parallel machines, so the methods of crossover and mutation should be suitable for use.

##### *Crossover*

Because the WSP with minimum makespan has the job due date problem, the chromosome may have a bad fitness value, an infeasible solution, through the traditional crossover operator. We provide a new crossover considering the time postponement concept to figure out the problem with due date restriction. The time postponement is the value that the non-expected event can delay for at most. The crossover operates two parents and creates a single offspring. It breeds the primary partitioning structure and better sub-schedules into offspring from one parent and then

fills the offspring with remaining genes derived from the other parent. The selection of sub-schedule is considering the job slackness, the time postponement, on each identical parallel machine. The crossover copies the better sub-schedules on some identical parallel machines from one parent to the offspring for preserving the good permutation of jobs. The other empty positions of the offspring can be filled with one way, which is a left-to-right scan from the other parent. We let  $C_{\nu k}$  be the completion time of job,  $d_{\nu k}$  be the due date of job on  $\nu$ th position of machine  $m_k$ ,  $n$  be the number of job consideration on machine  $m_k$ , and  $N_k$  be the number of jobs on machine  $m_k$ . Equation (38) indicates the slackness of job on  $\nu$ th position of machine  $m_k$ .  $SSL_{\nu k}$  of equation (39) means the sum of slackness values for  $n$  consecutive jobs on machine  $m_k$ , which are located from  $\nu$ th to  $(\nu+n-1)$ th position. After calculating the value  $SSL_{\nu k}$ , we can estimate the average value of slackness  $\overline{SSL}_{\nu k}$  for  $n$  jobs started from  $\nu$ th to  $(\nu+n-1)$ th position on machine  $m_k$ . The estimations of slackness values are defined as following:

$$SL_{\nu k} = d_{\nu k} - C_{\nu k}, \quad \nu = 1, 2, \dots, N_k; \quad k = 1, 2, \dots, K \quad (38)$$

$$SSL_{\nu k} = \sum_{\nu}^{\nu+n-1} SL_{\nu k}, \quad \nu = 1, 2, \dots, N_k; \quad 1 \leq n \leq N_k - \nu + 1; \quad k = 1, 2, \dots, K \quad (39)$$

$$\overline{SSL}_{\nu k} = SSL_{\nu k} \times n^{-1} \quad (40)$$

In the beginning of crossover, GA would choose one parent with better fitness value and breed the partitioning structure of the parent into the offspring as shown in Figure 3.

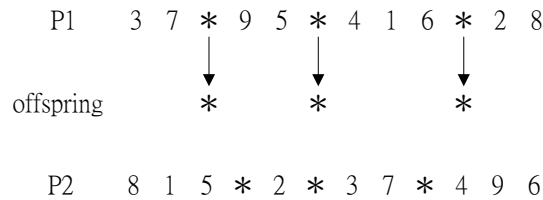


Figure 3. Copy the partitioning structure to offspring

Then for all jobs on each machine, GA calculates  $SSL_{\nu k}$  of all combinations of jobs in sequences and derives each  $\overline{SSL}_{\nu k}$  from each  $SSL_{\nu k}$ . Choose the smallest  $\overline{SSL}_{\nu k}$  for each machine and put the job combinations into the sub-schedules as

shown in Figure 4.

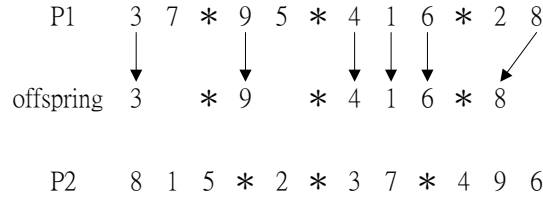


Figure 4. Copy the sub-schedules to offspring from the parent with better fitness value.

Finally, GA would use a left-to-right scan to fill the offspring with remaining genes derived from the other parent as shown in Figure 5.

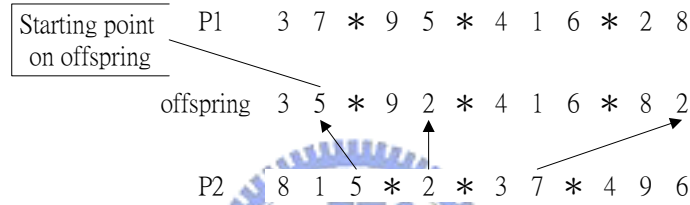


Figure 5. Fill the empty of the offspring from the other parent.

After recounting the execution of crossover, here is the procedure of crossover divided into three steps.

- Step1. Get the partitioning symbol \* from one parent which has better fitness value.
- Step2. Choose the sub-schedules from the parent with better fitness value. Let parameters  $k$  and  $\nu$  be one. The selection of sub-schedules is as following:
  - Step 2-1. For the  $\nu$  th position of machine  $m_k$  on the chromosome, calculate the slackness value  $SL_{\nu k}$  of job  $r_i$  on  $\nu$  th position.
  - Step 2-2. Let  $\nu = \nu + 1$ . If the value  $\nu$  is larger than  $N_k$ , go to step2-3. Otherwise, calculate the slackness value  $SL_{\nu k}$  of job on the  $\nu$  th position of machine  $m_k$  and go to step2-1.
  - Step 2-3. For  $\nu = 1, 2, \dots, N_k$  and  $1 \leq n \leq N_k - \nu + 1$ , estimate all kinds of average slackness value  $\overline{SSL}_{\nu n k}$ . Select the largest one  $\overline{SSL}_{\nu' n' k}$  among all  $\overline{SSL}_{\nu n k}$  and put the job combination from  $\nu'$  th position to  $\nu' + n' - 1$  th position on machine into the sub-schedule of machine  $m_k$ .
  - Step 2-3. Let  $k = k + 1$  and check the constraint of the number of available machines  $K$ . If  $k$  is larger than the number of available machines  $K$ , then go to step 2-4. Otherwise, Let index  $\nu$  be one and go to step2-1.

Step 2-4. Select the K sub-schedules of all machines and copy the K sub-schedules to offspring.

Step 3. Fill the empty of the offspring with the unscheduled genes by making a left-to-right scan from the other parent without violating the job due date restriction. The starting point of the filling can be generated randomly.

### Mutation

We use the swapping technique as the mutation method in this paper. The mutation proceeds by randomly choosing two genes on the chromosome and then swapping them. If the schedule shows infeasible after mutation technique, we would preserve the original one from due date violence. There are three possible exchanging ways through the swapping mutation: (1) the swapping of two jobs from the same identical parallel machine, (2) the swapping of two jobs from different identical parallel machines, (3) the swapping of one job and one partitioning symbol. Fig. 6(a), 6(b), and 6(c) shown below can express the swapping methods.

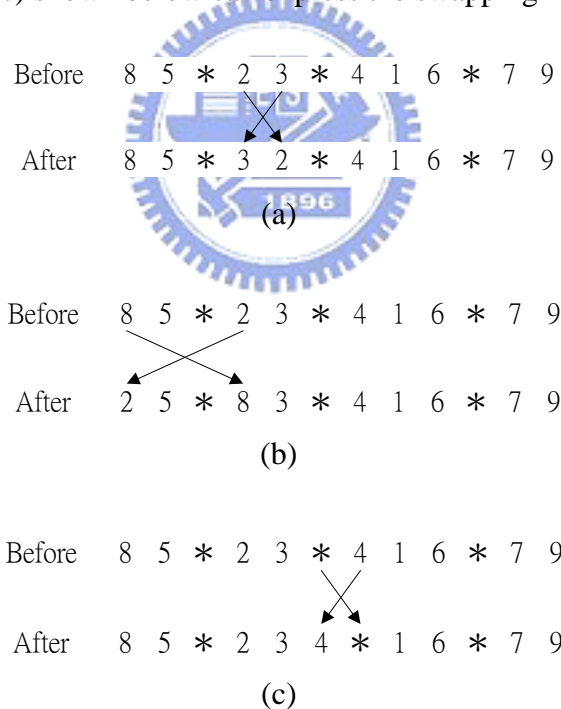


Figure 6. Illustration of swapping technique

### 4.2.5 Elitist Strategy

When the number of offspring in the pool is reaching to the expected level, we will mix the offspring with the original parents to get the enlarged population. Then we

use the roulette wheel as the concept of elitist strategy for choosing the better part of the enlarged population. That means the fitter chromosome is selected first for surviving to the next generation. In our GA cycle, the elitist way is to preserve the better chromosomes in each generation and reduce the errors of stochastic sampling. Through the elitist strategy, the number of chromosomes in each generation will be equal to the original population we determined in the beginning.

#### **4.2.6 Stopping Criteria**

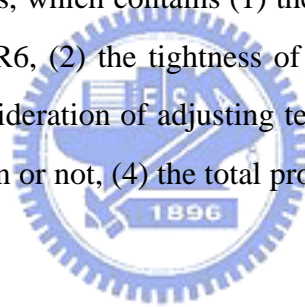
After the genetic operators and elitist strategy, the fitness of each chromosome in the population will be re-estimated by using the fitness function. There are several rules to decide if the GA cycle should be stopped: (1) see if the chromosomes in the current population are fitter than the ones in the previous population by calculating the total and average fitness values of all chromosomes in every cycle, (2) see if the best chromosome in the current GA pool is fitter than the best one in the old GA pool by calculating the fitness value of the best chromosome in every cycle, (3) see if the number of generation reaches to the level we requested. We choose the item (3) according to the convenience of GA operation. Therefore, in our experiment, the GA operation will be terminated if the number of generation reaches to what we prescribed.

### **5. Problem Design and Testing**

For the sake of comparing the performance of improving heuristics and the hybrid genetic algorithm, we design a set of 16 problems with different circumstances for testing. Each problem includes 25 parallel identical machines and 100 jobs, which are divided into 30 product types and should be completed before the given due date. The 100 jobs would be processed on the 25 parallel identical machines and each machine capacity is set to be three days, 4320 minutes. Here “minute” is used as the time unit for job processing time, job due dates, setup time, and machine capacity. In this paper, we highlight the impact of setup time of consecutive jobs from different product families or different operation temperatures. So the time cost by changing

probe card before the machine is ready to process the coming job with different product family is set to 80 or 120 minutes (80 or 120 minutes is according to different product family). The required times of adjusting temperature from room to high is set to be 60 minutes, from high to room is set to be 80 minutes, and from high to high is set to be 140 minutes. Because the time of adjusting temperature from room to room does not need to warm up or cool down the machine, it is set to be 0 minutes. The time of loading code before the machine is ready to process the coming job with different product type is set to be 5 minutes. And the initial setup time of machine from idle to processing state is set to be 100 minutes. The setup time of consecutive jobs from same product type is set to be 0 minutes under all operation temperatures.

The problem design is based on the wafer probing shop floor in an IC manufacturing factory of the Science-based Industrial Park, Taiwan. The problem test is divided into four factors, which contains (1) the product family ratio, including two grouping levels R2 and R6, (2) the tightness of due dates, including stable and increasing states, (3) the consideration of adjusting temperature, including setup time with temperature consideration or not, (4) the total processing time, including low and high levels.



#### Product Family Ratio ( $R$ )

The distribution of jobs to the product families is related to the setup time of consecutive jobs. We need to evaluate the influence of product families on the performance of scheduling solutions via the factor, product family ratio. If a product family has large number of jobs, it may lead to a smaller value of total setup time of scheduling solutions. Oppositely, if a product family has small number of jobs, it may result in a larger value of total setup time of machine schedules. Here we define an index, product family ratio, which is the division of the number of job product types by the number of job product families. There are 100 jobs divided into 30 product types in our test problem. For example, if the value of product family ratio is 2, it means that 30 product types of 100 jobs are distributed into 15 product families randomly. In our design, there are two levels for testing, R2 and R6, which means

30 product types of jobs are divided into 15 and 5 product families individually. The evaluation of product family ratio is expressed in equation (41).

$$\text{Product Family Ratio } (R) = \frac{\text{Number of product types}}{\text{Number of product families}} = \frac{J}{F} \quad (41)$$

Tightness of Due Dates ( $T\_Due$ )

Here we use tightness of due dates for evaluating the density of job due dates. It is including the job processing time, the expected setup time, the machine capacity before due dates, and the number of jobs with given due dates. The tightness index is defined as below:

$$\text{Tightness index } (TI(Y)) = \left\{ P(Y) + ES \times \frac{Num(Y)}{I} \right\} (K \times Cap(Y))^{-1}, Y = 1, 2, 3 \quad (42)$$

where the number of available machines  $K$  and the expected setup time  $ES$  are expressed in Section 2 and 3. Due dates of Jobs in the test problem are divided into three time points, which are 1, 2, and 3 days.  $P(Y)$  is denoted as the total processing time of jobs of which due dates are given before  $Y$ th due day point. We define the notation  $Cap(Y)$  as the available capacity of machine before  $Y$ th due day point. And  $Num(Y)$  is to express the number of jobs of which due dates are given before  $Y$ th due day point.

According to equation (42), we can evaluate three tightness indexes under three time points of due dates. If the tightness of due dates is stable, that means there are 30 jobs assigned for 1440 minutes of due dates, 35 jobs assigned for 2880 minutes of due dates, and 35 jobs assigned for 4320 minutes of due dates randomly. And the tightness values of due dates would be nearly equal. If the tightness of due dates is increasing, that means there are 5 jobs assigned for 1440 minutes of due dates, 15 jobs assigned for 2880 minutes of due dates, and 80 jobs assigned for 4320 minutes of due dates randomly. Besides, the tightness of due date 1440 minutes would be smaller than the tightness of due date 2880 minutes, and the tightness of due date 2880 minutes would be smaller than the tightness of due date 4320 minutes

### Temperature Consideration ( $T_e$ )

Because the setup time of loading temperature is longer than the setup time of peripheral hardware, we take the factor, temperature changing, into consideration in our problem design. The value of setup time is related to the product types, product families of two consecutive jobs generally. If temperature change of machine is considered in testing situation, it should be added in setup time. Our problem is designed to consider setup time with temperature changing or not.

### Total Processing Time ( $Total\_PT$ )

The value of total processing time would influence the degree of scheduling difficulties, so it is an index for evaluating the performance of scheduling heuristics. We generate two levels of total processing time, high and low, to represent the size of total machine workload. High and low levels of total processing time are set to be 54126 minutes and 66379 minutes, which have 1.5 and 1.85 days of machine utilization individually. Table 2 below shows the summary of 16 testing problems, and other related information suchlike product types, product families, tightness of due dates, and setup time of two consecutive jobs, is shown in the appendix.

Table 2. Summary of 16 problem design

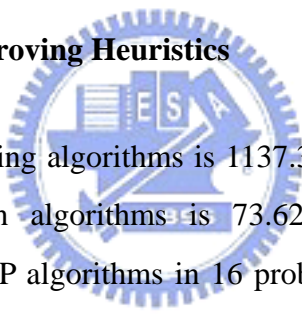
Problem No.	Product Family Ratio		Temperature Change		Tightness of due dates		Total processing time level		
	R=2	R=6	Yes	No	Increase	Stable	Low	High	total workload
1	●			●		●	●		54126
2	●			●		●		●	66379
3	●			●	●		●		54126
4	●			●	●			●	66379
5	●		●			●	●		54126
6	●		●			●		●	66379
7	●		●		●		●		54126
8	●		●		●			●	66379
9		●		●		●	●		54126
10		●		●		●		●	66379
11		●		●	●		●		54126
12		●		●	●			●	66379
13		●	●			●	●		54126
14		●	●			●		●	66379
15		●	●		●		●		54126
16		●	●		●			●	66379



## 6. Computational Results

Because our objective is minimum makespan, total setup time and the distribution of jobs to parallel machines both are related to the qualities of testing solutions. In order to find the setting parameters of improving heuristics for reducing total setup time, we put these WPSP algorithms into a lot of pre-tests. The following settings of WPSP algorithms are efficient in solving identical parallel-machine scheduling problems. The parameters  $A$  and  $B$  of modified sequential savings algorithm (MSSA) are set to be 0.975 and 0.55. The parameter  $\lambda$  of parallel insertion with the slackness (PIA II) is set to be 0.7. And the parameter  $\varphi$  of parallel insertion with the variance of regret measure (PIA IV) is set to be 0.8. The improving heuristics are encoded in Visual Basic 6.0, which are implemented in the compiled form on a PC with AMD 1150 MHz CPU and 512 MB RAM.

### 6.1 ANOVA Analysis of Improving Heuristics



The CPU times cost by saving algorithms is 1137.396 seconds in average, and the CPU time cost by insertion algorithms is 73.629 seconds in average. The computational results of WPSP algorithms in 16 problems are expressed in Table 3. The testing results show that SSA and SIA are not robust in our testing problems because SSA and SIA would generate unfeasible solutions in some cases. Therefore, we look for best solutions exclusive of SSA and SIA and find that PIA III has the largest number of best solutions via Table 3. And the average of testing solutions solved by PIA III in 16 problems is the smallest, which means PIA III is most efficient in solving the 16 testing problems of WPSP with minimum makespan. By considering one experimental factor once, the computational results of 16 problems can be transformed into the performance comparison of all situations as shown in Table 4. From the opinion of comparing mean and standard deviation of solutions in all kinds of situations, PIA III having 8 and 7 smallest ones individually also shows that it outperforms other WPSP algorithms except for the situation, where total processing time is low.

Table 3. Computational results of WPSP algorithms in 16 test problems.

Problem No.	Expected Capacity	SSA	MSSA	SIA	PIA	PIA I	PIA II	PIA III	PIA IV
		Cmax	Cmax	Cmax	Cmax	Cmax	Cmax	Cmax	Cmax
1	2730	2965	2724	2837	2660	2599*	2600	2639	2653
2	3297		3709		3240	3194	3218	3157*	3291
3	2730	2661	2611*	2611	2703	2616	2632	2611*	2669
4	3297	3186	3083*	3116	3197	3115	3158	3105	3140
5	3196	3152	2795	2837	2879	2728*	2757	2730	2778
6	3763		4308		3461	3265	3357	3248*	3955
7	3196	2682	2739	2611	2805	2683	2758	2659*	2770
8	3763	3253	3223	3116	3376	3201	3223	3194*	3242
9	2687	2692	2697	2664	2635	2566*	2569	2594	2644
10	3253	3648	3476		3193	3140	3111*	3125	3225
11	2687	2548	2597	2604	2627	2577	2624	2565*	2624
12	3253	3113	3127	3063	3094	3081	3066*	3083	3116
13	3153	3600	2766	2977	2782	2732	2714*	2731	2930
14	3719		3596		3447	3284	3274*	3274*	3675
15	3153	2721	2713	2696	2728	2662	2669	2654*	2729
16	3719	3201	3235	3245	3295	3223*	3228	3223*	3269
Mean			3087.438		3007.625	2916.625	2934.875	2912	3044.375
No. of *			2			4	4	9	

Result with grey background indicates a unfeasible solution in scene. Label \* means the best of all exclusive of SSA and SIA.

Table 4. Computational results of improving heuristics under all kinds of situations.

	n	SSA		MSSA		SIA		PIA	
		Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev
Total	16			3087.44	485.11			3007.63	309.04
R=2	8			3149.00	589.21			3040.13	314.85
R=6	8			3025.88	385.29			2975.13	321.12
Te=Yes	8			3171.88	558.31			3096.63	325.18
Te=No	8			3003.00	419.85			2918.63	284.27
T_Due=Stable	8			3258.88	600.47			3037.13	339.57
T_Due=Increase	8	2920.63	292.66	2916.00	276.65	2882.75*	275.89	2978.13	295.58
Total_PT=Low	8	2877.63	350.02	2705.25	69.66	2729.63	138.16	2727.38	89.24
Total_PT=High	8			3469.63	406.89			3287.88	131.07
Number of *						1			
		PIAI		PIAII		PIAIII		PIAIV	
	n	Mean	Stdev	Mean	Stdev	Mean	Stdev	Mean	Stdev
Total	16	2916.63	287.85	2934.88	289.49	2912.00*	279.90	3044.38	392.77
R=2	8	2925.13	292.62	2962.88	305.12	2917.87*	280.73	3062.25	443.37
R=6	8	2908.13	302.93	2906.88	291.02	2906.13*	298.32	3026.50	365.06
Te=Yes	8	2972.25	291.64	2997.50	295.96	2964.13*	291.52	3168.50	455.44
Te=No	8	2861.00	292.28	2872.25	288.14	2859.88*	276.92	2920.25	296.54
T_Due=Stable	8	2938.50	310.03	2950.00	322.71	2937.25*	289.28	3143.88	483.84
T_Due=Increase	8	2894.75	283.45	2919.75	273.71	2886.75	287.65	2944.88	271.75
Total_PT=Low	8	2645.37*	65.48	2665.38	71.45	2647.88	59.79	2724.63	101.39
Total_PT=High	8	3187.88	71.37	3204.38	92.00	3176.13*	69.67	3364.13	294.29
Number of *		1				7			

Result with grey background means there are unfeasible solutions in scenario. Label \* indicates the best among algorithms in situations considering single factor and whole conditions.

In order to find the effects of improving heuristics and experimental factors on the problem design, we use statistical analysis by applying statistical software, SAS. First of all, we check the satisfaction of normality assumption for 96 data of Table 3 except for SSA and SIA. The check of normality assumption is expressed in Table 5 and the solutions are normally distributed. Then use ANOVA to check for the significances of all experimental factors and interactions. The summary of ANOVA table shown in Table 6 shows that five single factors, product family ratio, temperature changing consideration, tightness of due date, total processing time level,

and algorithms, would significantly affect the solutions of WPSP with minimum makespan under 99% confidential intervals. Besides,  $p$  values of interactions less than 0.01 also have significant effect on the performance of testing problems. Through Duncan's multiple comparison as shown in Table 7, the statistical results show that the multiple comparisons divide WPSP algorithms into two groups, A and B. The same letter of Duncan's groups indicates that there is no significant difference between WPSP algorithms. So the first group is MSSA, PIA IV, and PIA, of which the performance of solutions is inferior to the second group of PIA II, PIA I, and PIA III.

Table 5. Check of normality assumption for 96 solutions in 16 test problems.

Test	Statistic	p Value		
Shapiro-Wilk	W	0.886611	Pr < W	<0.0001
Kolmogorov-Smirnov	D	0.17839	Pr > D	<0.0100
Cramer-von Mises	W-Sq	0.58319	Pr > W-Sq	<0.0050
Anderson-Darling	A-Sq	3.464467	Pr > A-Sq	<0.0050

Table 6. The summary of ANOVA table under 99% confidential intervals.

Factor	SS	d.f.	MS	F value	p-value
R	63500	1	63500	10.548	<0.01
Te	583908	1	583908	96.991	<0.01
T_Due	350779	1	350779	58.267	<0.01
Total_PT	8516246	1	8516246	1414.611	<0.01
Algorithm	432625	5	86525	14.372	<0.01
R*Te	184	1	184	0.031	
R*T_Due	11726	1	11726	1.948	
Te*T_Due	47126	1	47126	7.828	<0.01
R*Total_PT	13325	1	13325	2.213	
Te*Total_PT	32893	1	32893	5.464	
T_Due*Total_PT	170775	1	170775	28.367	<0.01
R*Algorithm	33404	5	6681	1.110	
Te*Algorithm	59144	5	11829	1.965	
T_Due*Algorithm	313319	5	62664	10.409	<0.01
Total_PT*Algorithm	168813	5	33763	5.608	<0.01
Error	156525.3	26	6020		
$F_{0.01,1,26} = 7.72$ $F_{0.01,5,26} = 3.82$					

Table 7. Duncan's multiple comparisons for the performance of WPSP algorithms.

Duncan Grouping	Mean	No. of problems	Algorithm
A	3087.44	16	MSSA
A	3044.38	16	PIA IV
A	3007.63	16	PIA
B	2934.88	16	PIA II
B	2916.63	16	PIA I
B	2912	16	PIA III

## 6.2 Computational Results of GA with Initial Population from WPSP algorithms

We use WPSP algorithms except for SSA and SIA for generating initial population of GA. Because the initial solutions are not sufficient enough, we make use of these initial solutions to enlarge our population size for larger species of chromosomes. In our problem design, we set the population size to be 30. Other genetic factors, mutation rate and generation size, are considered in testing problems because they would affect genetic combinations of chromosomes. We select problems No.7 and No. 8 of Table 2 for testing the performance and solution time of GA. One is the situation that product family ratio is 2, temperature changing is considered, total processing time level is low, and tightness of due dates is increasing. The other selected is the situation that product family ratio is 2, temperature changing is considered, total processing time level is high, and tightness of due dates is increasing. The mutation rate (denoted as  $pm$ ) is divided into five levels, 0, 0.25, 0.5, 0.75, and 1. And the hybrid GA is proceeding until the number of generation (denoted as  $gen$ ) is equal to 1500. Each problem is solved by hybrid GA with different mutation rates and repeated four times for checking if the mutation rate is significantly effective.

The statistical results of hybrid GA in problem No. 7 and No. 8 are shown in Table 8. It shows that hybrid GA would improve the initial solutions while the mutation rate is larger than 0. And we know that the generation number is proportional to the performance of scheduling solutions of WPSP with minimum makespan. The running times of GA in 250, 500, and 750 generations are about 191 seconds, 401 seconds, and 563 seconds individually. They are apparently larger than the running time of improving heuristics. Because the roulette wheel method is selecting chromosomes based on fitness values randomly, there is not a definite mutation rate used for finding the best solution of hybrid GA. So we consider all kinds of mutation rates in generations and plot the flowcharts of solutions solved by hybrid GA as shown in Figure 7 to Figure 10 in the appendix. They show the trend of mean performance solved by hybrid GA in problem No. 7 and No. 8. Observing the tendency of solutions solved in Figure 7 and Figure 9, it reveals that the value of

average performance via hybrid GA would drop very fast before initial generations (about 100 generations). Based on statistical data of hybrid GA repeated four times in 1500 generations, we find that the best solution of hybrid GA is tending to be improved after later generations (about 40 generations) when the total processing time level is low. In contrast with the situation while the total processing time level is high, we can see that the improving time point of hybrid GA in problem No. 7 is significantly later. So hybrid GA may have a faster speed of feedback for improving the best solution under tough situations.

Table 8. The comparison of hybrid GA with WPSP algorithms and improving heuristics in testing problems.

Problem No.	Iteration	Gen	Hybrid GA		CPU Time (sec) in average	The best sol. by Improving Heuristics	CPU Time (sec) of the best sol.
			Cmax	pm		Cmax	
7	1	250	3046	0.5	191.35	2659*	75.64*
		500	3000	0.5	401.82		
		750	2977	0.5	563.71		
		1500	2977	0.5	1186.13		
7	2	250	3052	0.75	188.11	2659*	75.64*
		500	3030	0.75	413.98		
		750	2985	0.75	530.26		
		1500	2966	0.75	1199.86		
7	3	250	3116	1	200.91	2659*	75.64*
		500	3024	1	411.12		
		750	3010	1	580.32		
		1500	2911	0.75	1210.79		
7	4	250	3073	0.5	194.46	2659*	75.64*
		500	2999	0.75	409.61		
		750	2999	0.75	549.53		
		1500	2927	0.5	1203.72		
8	1	250	3675	0.25	200.91	3194*	73.21*
		500	3659	0.25	399.57		
		750	3650	0.75	581.27		
		1500	3608	0.75	1175.23		
8	2	250	3635	1	192.82	3194*	73.21*
		500	3629	1	391.13		
		750	3608	0.75	577.21		
		1500	3608	0.75	1189.11		
8	3	250	3667	0.75	196.41	3194*	73.21*
		500	3650	0.75	388.45		
		750	3650	0.75	562.13		
		1500	3635	0.25	1179.92		
8	4	250	3675	1	213.61	3194*	73.21*
		500	3659	0.75	400.01		
		750	3637	1	580.66		
		1500	3579	1	1180.36		

Label \* means the better between hybrid GA and improving heuristics.

### 6.3 Further Improvement of Hybrid GA with Initial Population from Improving Heuristics

We apply scheduling solutions of improving heuristics in initial population of hybrid GA for further improvement in 16 testing problems. We run 1500 generations of hybrid GA with 0.5, 0.75, and 1 of mutation rate for confirming

whether the scheduling solutions of improving heuristics can be improved or not. The computational results are shown in Table 9 and expresses that we may not use hybrid GA for improving solutions of improving heuristics in the situation where the tightness of due date is stable. And hybrid GA can improve solutions generated by improving heuristics in problem No. 8 and No. 16, where the temperature change consideration is yes, tightness of due date is increasing, and total processing time level is high. The trends of solutions generated by hybrid GA in problem No. 8 and No. 16 are shown in Figure 11 and Figure 12 individually. The evidential results show that hybrid GA can improve in earlier generations in problem No. 8 (R=2, Te=yes, T\_Due=increase, and Total\_PT=high) than hybrid GA in problem No. 16 (R=6, Te=yes, T\_Due=increase, and Total\_PT=high). Through solutions of hybrid GA with initial population generated by WPSP algorithms and improving heuristics, we find that the performance of hybrid GA would stop improving after latter periods of generations (about 1500 generations).

Table 9. Computation results of GA with initialization of improving heuristics.

Problem No.	Improving Heuristics	Hybrid GA	Problem No.	Improving Heuristics	Hybrid GA
	Cmax	Cmax		Cmax	Cmax
1	2599	2599	9	2566	2566
2	3157		10	3111	
3	2611	2611	11	2565	2565
4	3083	3083	12	3066	3066
5	2728	2728	13	2714	
6	3248		14	3274	
7	2659	2659	15	2654	2654
8	3194	3160*	16	3223	3183*
No. of *		1	No. of *		1

Result with grey background indicates GA can't generate enough feasible strings as population.  
Label \* indicates hybrid GA can find better solutions compared with improving heuristics.

## 7. Conclusion

The wafer probing scheduling problem (WPSP) is a practical version of the parallel-machine scheduling problem, which has many real-world applications including the integrated circuit (IC) manufacturing industry and other industries containing the manufacturing process with parallel machines. In this paper, we consider WPSP with the objective to minimize the maximum completion time and

formulate the WPSP with minimum makespan as an integer-programming problem. To solve the WPSP with minimum makespan effectively, we proposed improving heuristics and the hybrid GA for our cases. The computational results show that improving heuristics and hybrid GA are efficient tools for solving our testing problems of WPSP with minimum makespan. And GA with initial population by improving heuristics can make scheduling solutions outperform scheduling ones of improving heuristics. From now on, the collection of initial population satisfying the WPSP with minimum makespan is our studying point because it not only expands the variety of genetic composition but affects the average and best solutions of GA.



## Reference

- [1] Pearn, W.L., Chung, S.H., and Yang, M.H., "Minimizing The Total Machine Work Load For The Wafer Probing Scheduling Problem (WPSP)," IIE transactions, 34, 211-220 (2002).
- [2] Pearn, W.L., Chung, S.H., and Yang, M.H., "The Wafer Probing Scheduling Problem (WPSP)," Journal of the Operational Research Society, 53, 864-874 (2002).
- [3] Sethi, R., "On The Complexity Of Mean Flow Time Scheduling," Mathematics Of Operations Research, 2(4), 320-330 (1977).
- [4] Garey, M.R. and Johnson, D.S., "Computer And Intractability: A Guide To The Theory Of NP-Completeness," San Francisco: W H Freeman. (1979).
- [5] Potts C.N., "Analysis Of A Linear Programming Heuristic For Scheduling Unrelated Parallel Machines," Discrete Applied Mathematics, 10(2), 155-164 (1983).
- [6] Bernstein, D., Pinter, R.Y., and Rodeh, M., "Optimal Scheduling Of Arithmetic Operations In Parallel With Memory," The Annual ACM Symposium On Principles Of Programming Languages, New York (1985).
- [7] Luh, P.B., Hoitomt, D.J., and Max, E., "Parallel Machine Scheduling Using Lagrangian Relaxation," IEEE International Conference On computer Integrated Manufacturing, New York, 244-248 (1988).
- [8] Narahari, Y. and Srigopal, R., "Real-world Extension To Scheduling Algorithms Based On Lagrangian Relaxation," Proceedings In Engineering Sciences 21st, 415-433 (1996).
- [9] Cheng, T.C.E. and Sin, C.C.S., "State-of-the –art Review Of Parallel –machine Scheduling Research," European Journal Of Operational Research, 47(3), 271-292 (1990).
- [10] Min, L. and Cheng W., "A Genetic Algorithm For Minimizing Makespan In The



Case Of Scheduling Identical Parallel Machines,” *Artificial Intelligence Engineering*, 13, 399-403 (1999).

[11] Gupta, J.N.D. and Ruiz-Torres, J., “A Listfit Heuristic For Minimizing Makespan On Identical Parallel Machines,” *Production Planning & Control*, 12(1), 28-36 (2001).

[12] Azizoglu, M. and Kirca, O., “Tradiness Minimization On Parallel Machines,” *International Journal Of Production Economics*, 55, 163-168 (1998).

[13] Lee, Y.H. and Pinedo M., “Scheduling Jobs On Parallel Machines With Sequence-Dependent Setup Times,” *European Journal Of Operational Research*, 100, 464-474 (1997).

[14] Park, Y.G, Kim, S.Y., and Lee, Y.H., “Scheduling Jobs On Parallel Machines Applying Neural Network And Heuristic Rules,” *Computers & Industrial Engineering*, 38, 189-202 (2000).

[15] Hurkens, C.A.J. and Vredeveld, T., “Local Search For Multiprocessor Scheduling: How Many Moves Does It Take To a Local Optimum,” *Operations Research Letters*, 31, 137-141 (2003).

[16] Veen, J.A.A.V.D and Zhang, S.H., “Low-Complexity Algorithm For Sequencing Jobs With A Fixed Number Of Job-Classes,” *Computers & Operations Research*, 23(11), 1059-1067 (1996).

[17] Clark, G. and Wright, J., “Scheduling Vehicles from A Central Depot To A Number Of Delivery Points,” *Operation Research*, 12, 568 (1964).

[18] Golden, B., “Evaluate A Sequential Vehicle Routing Algorithm,” *AIIE Trans*, 9, 204-208 (1977).

[19] Pearn, W.L., Chung, S.H., Yang, M.H., and Chen Y.H., “Algorithms for the Wafer Probing Scheduling Problem with Sequence Dependent Setup Time and Due Date Restriction,” submitted to *Journal of the Operational Research*, (2003).

- [20] Solomon, M.M., "Algorithms For The Vehicle Routing And Scheduling Problem With Time Window Constraints," *Operation Research*, 35(2), 254-265 (1987).
- [21] Potvin, Y. and Rousseau, J.M., "A Parallel Route Building Algorithm For The Vehicle Routing And Scheduling Problem With Time Windows," *European Journal Of Operation Research*, 66, 331-340, (1993).
- [22] Pearn, W.L., Chung, S.H., Yang, M.H., and Shiao K.P., "Parallel Insertion Algorithms for the Wafer Probing Scheduling Problem," submitted to *European Journal of Operational Research*, (2003).
- [23] Zomaya, A.Y. and The, Y.H., "Observation on Using Genetic Algorithms for Dynamic Load-Balancing," *IEEE*, 12(9), 899-911, (2001).
- [24] Cheng, R., Gen, M., and Tosawa, T., "Minmax Earliness/Tardiness Scheduling In Identical Parallel Machine System Using Genetic Algorithms," *Computers ind. Engng*, 29(1-4), 513-517 (1995).
- [25] Min, L. and Cheng, W., "A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines," *Artificial Intelligence in Engineering*, 13, 399-403, (1999).
- [26] Cochran, J.K., Horng, S.M., and Fowler, J.W., "A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines," *Computers & Operations Research*, 30, 1087-1102, (2003).
- [27] Cheng, R. and Gen, M., "Parallel Machine Scheduling Problems Using Memetic Algorithms," *Computers ind. Engng*, 33(3-4), 761-764, (1997).
- [28] Serifoglu, F.S. and Ulusoy, G., "Parallel machine scheduling with earliness and tardiness penalties," *Computers & Operations Research*, 26, 773-787, (1999).
- [29] Herrmann, J.W., "A Genetic Algorithm for Minimax Optimization Problems," *Proceedings of the Congress of Evolutionary Computation*, 1099-1103, (1999).
- [30] Tamaki, H., Nishino, E., and Abe, S., "A Genetic Algorithm Approach to

Multi-Objective Scheduling Problems with Earliness and Tardiness Penalties,”  
Proceedings of the Congress of Evolutionary Computation, 46-52, (1999).

[31] Viginer, A., Sonntag, B., and Portmann, M-C., “A hybrid method for a  
parallel-machine scheduling problem,” International Conference on Emerging  
Technologies and Factory Automation, 671-678, (1999).

[32] Lin, C.C., “A Genetic Algorithm for Unrelated Parallel-Machine Scheduling  
Problems,” Master. Thesis, Chaoyang University of Technology, Taichung,  
(2001).



## Appendix

Table A1. Processing times of jobs with product types and product families under low total processing time level.

Product Type	Processing Time	Product Family(R6)	Product Family(R2)	Testing Temperature	Product Type	Processing Time	Product Family(R6)	Product Family(R2)	Testing Temperature
1	404	3	7	High	16	429	5	4	High
2	593	1	10	Room	17	623	1	2	High
3	411	2	2	High	18	417	4	12	High
4	420	5	11	High	19	440	4	12	High
5	681	4	9	High	20	434	4	14	High
6	606	3	1	Room	21	680	4	7	Room
7	585	3	1	High	22	430	1	3	High
8	403	1	6	Room	23	694	2	13	Room
9	495	2	2	Room	24	455	5	1	Room
10	677	5	10	High	25	451	1	3	Room
11	517	4	3	Room	26	538	4	2	High
12	519	3	1	High	27	663	4	5	High
13	662	4	6	Room	28	570	3	10	High
14	499	2	14	High	29	618	3	7	Room
15	660	3	6	Room	30	441	2	10	Room

Table A2. Processing times of jobs with product types and product families under high total processing time level.

Product Type	Processing Time	Product Family(R6)	Product Family(R2)	Testing Temperature	Product Type	Processing Time	Product Family(R6)	Product Family(R2)	Testing Temperature
1	780	4	5	High	16	522	5	10	Room
2	752	4	14	Room	17	760	1	8	Room
3	723	4	4	Room	18	712	3	5	High
4	681	3	4	Room	19	588	3	2	High
5	686	2	1	Room	20	751	1	8	Room
6	670	1	12	Room	21	740	2	15	Room
7	582	2	13	High	22	629	5	3	Room
8	686	1	14	Room	23	747	1	11	Room
9	742	5	6	High	24	756	4	4	Room
10	682	3	3	Room	25	717	3	2	Room
11	536	5	14	Room	26	642	2	9	High
12	707	2	1	High	27	621	2	10	High
13	594	4	4	High	28	777	5	5	Room
14	790	1	3	Room	29	569	2	1	High
15	505	2	7	Room	30	750	5	8	High

Table A3. Tightness of due dates in 16 testing problems.

Problem No.	Expected Setup Time			Total Processing Time			Available Capacity			Tightness of Due Dates		
	Due dates of jobs			Due dates of jobs			Due dates of jobs			Due dates of jobs		
	1440	2880	4320	1440	2880	4320	1440	2880	4320	1440	2880	4320
1	169	198	198	16066	34459	54126	36000	72000	108000	45.10%	48.13%	50.30%
2	192	225	225	19157	42402	66379	36000	72000	108000	53.75%	59.20%	61.67%
3	28	85	452	2913	10973	54126	36000	72000	108000	8.17%	15.36%	50.54%
4	32	96	514	3416	13404	66379	36000	72000	108000	9.58%	18.75%	61.94%
5	309	361	361	16066	34459	54126	36000	72000	108000	45.49%	48.36%	50.45%
6	332	388	388	19157	42402	66379	36000	72000	108000	54.14%	59.43%	61.82%
7	51	155	825	2913	10973	54126	36000	72000	108000	8.23%	15.46%	50.88%
8	55	166	887	3416	13404	66379	36000	72000	108000	9.64%	18.85%	62.28%
9	156	183	183	16066	34459	54126	36000	72000	108000	45.06%	48.11%	50.29%
10	180	209	209	19157	42402	66379	36000	72000	108000	53.71%	59.18%	61.66%
11	26	78	418	2913	10973	54126	36000	72000	108000	8.16%	15.35%	50.50%
12	30	89	479	3416	13404	66379	36000	72000	108000	9.57%	18.74%	61.91%
13	296	346	346	16066	34459	54126	36000	72000	108000	45.45%	48.34%	50.44%
14	320	372	372	19157	42402	66379	36000	72000	108000	54.10%	59.41%	61.81%
15	49	148	791	16066	34459	54126	36000	72000	108000	8.23%	15.45%	50.85%
16	53	160	851	3416	13404	66379	36000	72000	108000	9.64%	18.84%	62.25%

Table A4. All jobs with product types and due dates while tightness of due dates is stable and total processing time level is low.

Job ID	Product Type	Due Date	Job ID	Product Type	Due Date	Job ID	Product Type	Due Date	Job ID	Product Type	Due Date	Job ID	Product Type	Due Date
1	1	4320	21	21	4320	41	4	4320	61	27	1440	81	14	1440
2	2	2880	22	22	4320	42	5	2880	62	14	1440	82	15	4320
3	3	4320	23	23	1440	43	20	2880	63	15	2880	83	10	4320
4	4	2880	24	24	2880	44	5	4320	64	3	4320	84	25	1440
5	5	4320	25	25	2880	45	11	2880	65	15	1440	85	4	4320
6	6	1440	26	26	4320	46	8	2880	66	18	2880	86	29	4320
7	7	4320	27	27	4320	47	18	2880	67	21	4320	87	20	4320
8	8	2880	28	28	2880	48	17	4320	68	15	2880	88	10	1440
9	9	2880	29	29	4320	49	30	1440	69	3	4320	89	11	1440
10	10	2880	30	30	1440	50	12	4320	70	8	2880	90	23	1440
11	11	4320	31	1	1440	51	27	4320	71	9	2880	91	9	1440
12	12	2880	32	14	1440	52	22	2880	72	8	2880	92	10	1440
13	13	4320	33	7	4320	53	20	1440	73	11	1440	93	18	1440
14	14	2880	34	29	2880	54	17	2880	74	24	1440	94	16	1440
15	15	4320	35	15	2880	55	27	2880	75	28	4320	95	6	1440
16	16	4320	36	27	2880	56	5	1440	76	2	4320	96	14	1440
17	17	4320	37	27	2880	57	18	2880	77	20	4320	97	29	1440
18	18	1440	38	13	2880	58	10	4320	78	27	4320	98	29	1440
19	19	2880	39	2	2880	59	16	2880	79	2	4320	99	14	1440
20	20	4320	40	12	2880	60	30	2880	80	12	1440	100	19	1440

Table A5. All jobs with product types and due dates while tightness of due dates is increasing and total processing time level is low.

Job ID	Product Type	Due Date	Job ID	Product Type	Due Date	Job ID	Product Type	Due Date	Job ID	Product Type	Due Date	Job ID	Product Type	Due Date
1	1	4320	21	21	4320	41	4	4320	61	27	2880	81	14	4320
2	2	4320	22	22	4320	42	5	4320	62	14	2880	82	15	2880
3	3	4320	23	23	4320	43	20	4320	63	15	4320	83	10	4320
4	4	4320	24	24	2880	44	5	4320	64	3	4320	84	25	2880
5	5	4320	25	25	4320	45	11	4320	65	15	4320	85	4	4320
6	6	1440	26	26	4320	46	8	4320	66	18	4320	86	29	4320
7	7	2880	27	27	1440	47	18	2880	67	21	4320	87	20	4320
8	8	4320	28	28	4320	48	17	4320	68	15	4320	88	10	4320
9	9	4320	29	29	4320	49	30	4320	69	3	2880	89	11	4320
10	10	4320	30	30	2880	50	12	4320	70	8	4320	90	23	4320
11	11	4320	31	1	4320	51	27	2880	71	9	4320	91	9	4320
12	12	4320	32	14	4320	52	22	4320	72	8	4320	92	10	4320
13	13	4320	33	7	1440	53	20	4320	73	11	4320	93	18	4320
14	14	4320	34	29	1440	54	17	4320	74	24	4320	94	16	4320
15	15	4320	35	15	4320	55	27	2880	75	28	4320	95	6	4320
16	16	4320	36	27	4320	56	5	4320	76	2	2880	96	14	4320
17	17	2880	37	27	4320	57	18	2880	77	20	4320	97	29	4320
18	18	4320	38	13	4320	58	10	4320	78	27	4320	98	29	4320
19	19	4320	39	2	4320	59	16	4320	79	2	4320	99	14	4320
20	20	4320	40	12	2880	60	30	1440	80	12	4320	100	19	4320

Table A6. All jobs with product types and due dates while tightness of due dates is stable and total processing time level is high.

Job ID	Product Type	Due Date	Job ID	Product Type	Due Date	Job ID	Product Type	Due Date	Job ID	Product Type	Due Date	Job ID	Product Type	Due Date
1	1	2880	21	21	4320	41	8	4320	61	8	1440	81	24	1440
2	2	4320	22	22	2880	42	10	4320	62	6	2880	82	18	1440
3	3	4320	23	23	4320	43	18	4320	63	19	1440	83	6	2880
4	4	4320	24	24	2880	44	24	1440	64	2	2880	84	11	2880
5	5	4320	25	25	2880	45	22	4320	65	16	1440	85	12	2880
6	6	2880	26	26	4320	46	30	2880	66	12	1440	86	27	1440
7	7	4320	27	27	1440	47	23	1440	67	9	4320	87	29	2880
8	8	4320	28	28	2880	48	15	1440	68	16	2880	88	13	4320
9	9	1440	29	29	4320	49	30	4320	69	12	2880	89	17	1440
10	10	4320	30	30	4320	50	13	2880	70	20	1440	90	21	2880
11	11	1440	31	26	4320	51	13	2880	71	3	2880	91	8	2880
12	12	1440	32	27	4320	52	29	1440	72	26	2880	92	16	1440
13	13	1440	33	23	4320	53	4	2880	73	1	4320	93	16	2880
14	14	2880	34	7	4320	54	14	4320	74	15	2880	94	11	1440
15	15	1440	35	16	2880	55	6	2880	75	12	4320	95	13	1440
16	16	1440	36	28	4320	56	8	2880	76	29	1440	96	14	2880
17	17	4320	37	18	1440	57	23	1440	77	16	1440	97	10	2880
18	18	4320	38	8	4320	58	8	4320	78	5	2880	98	24	1440
19	19	4320	39	9	2880	59	15	2880	79	16	2880	99	3	1440
20	20	2880	40	21	4320	60	11	4320	80	19	4320	100	29	1440









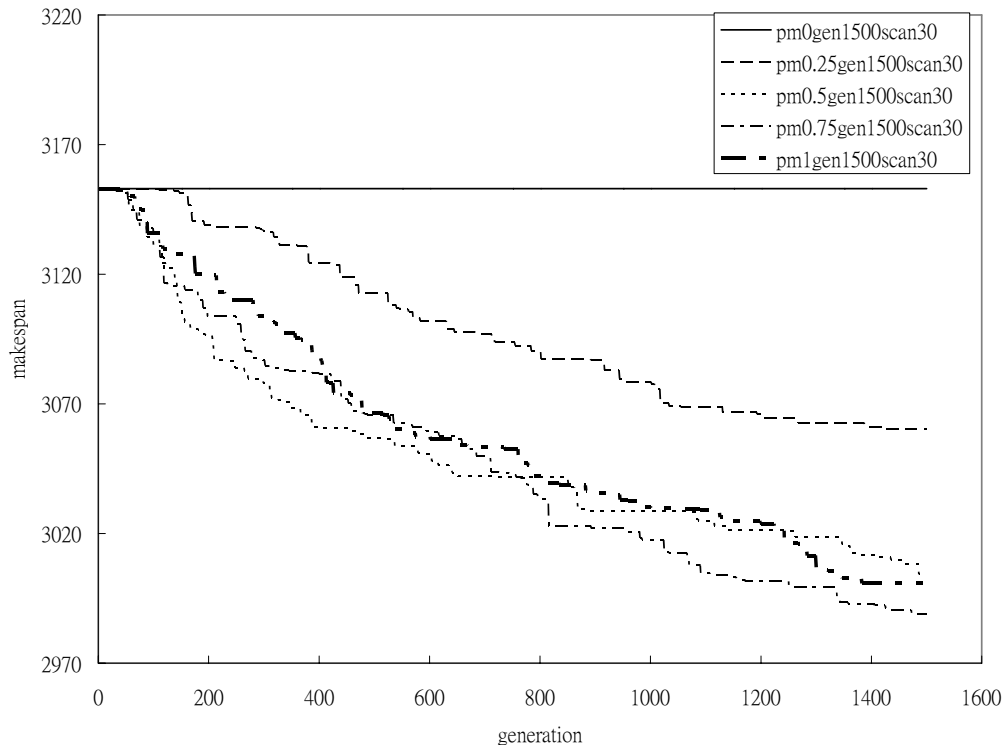


Figure 8. The trend of best solution in population of hybrid GA repeated four times in problem No. 7.

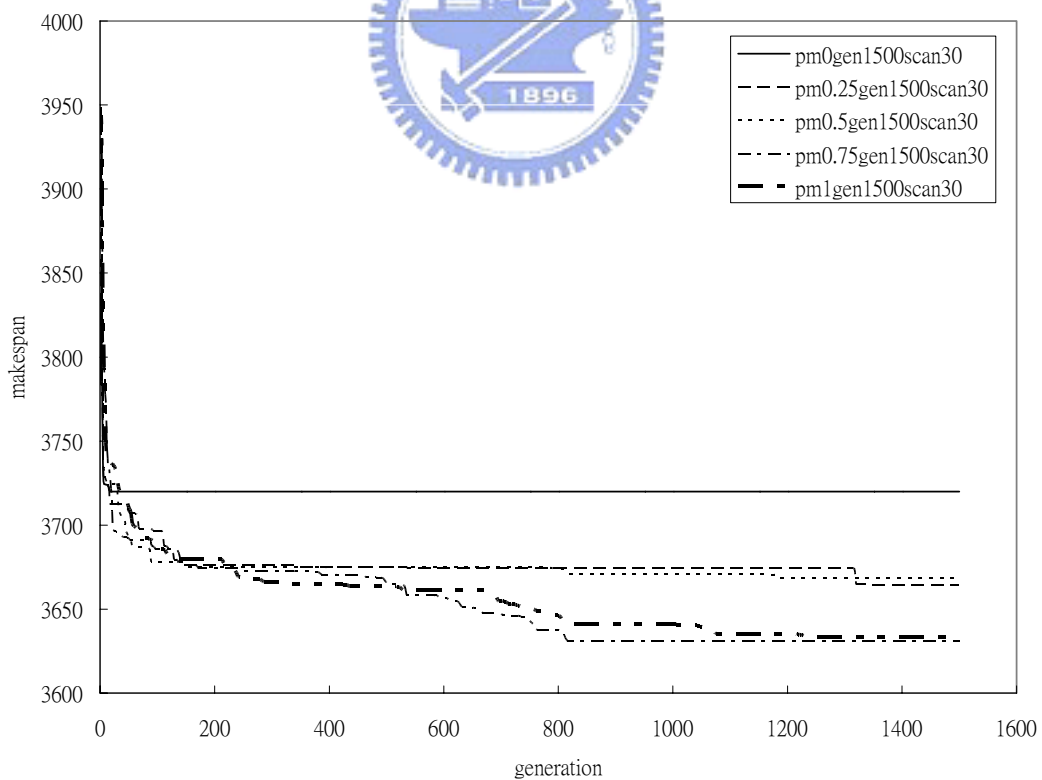


Figure 9. The trend of average solutions in population of hybrid GA repeated four times in problem No. 8.

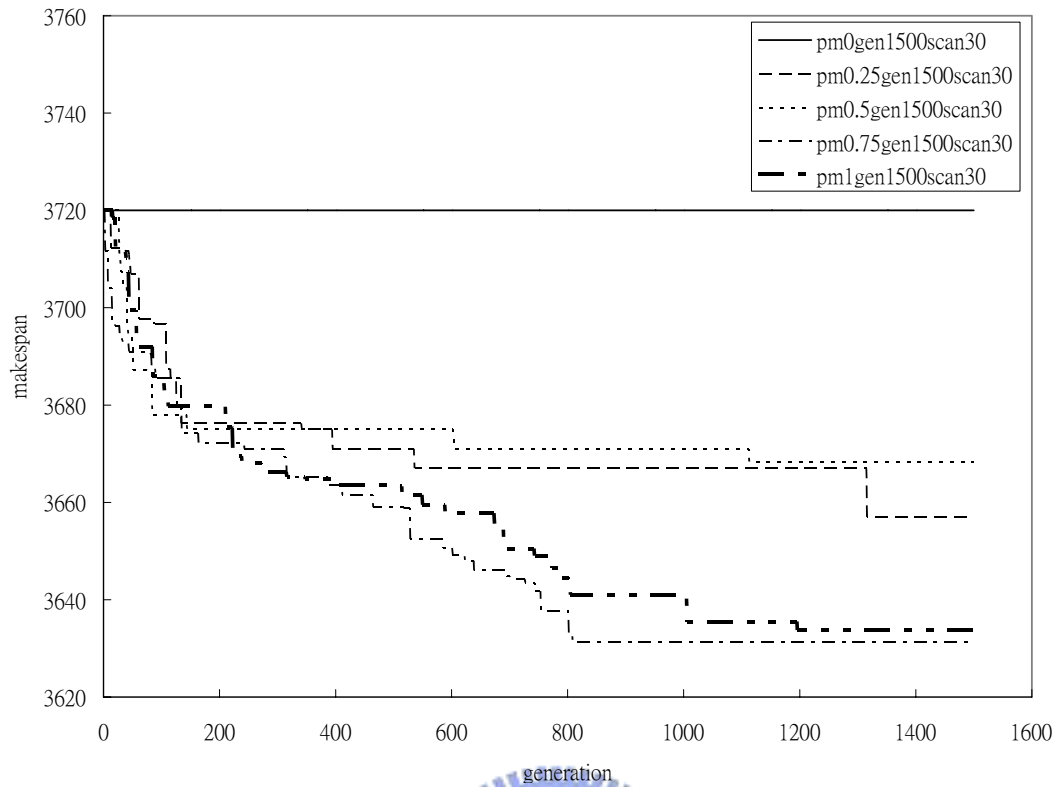


Figure 10. The trend of average solutions in population of hybrid GA repeated four times in problem No. 8.

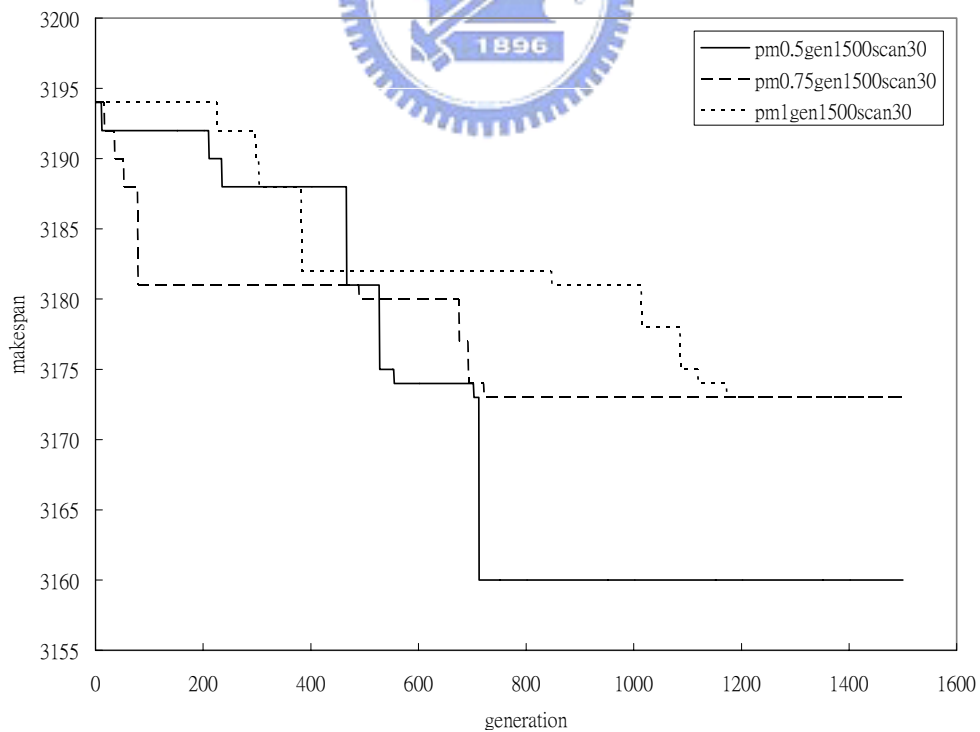


Figure 11. The further improvement of hybrid GA with initialization by improving heuristics in problem No. 8.

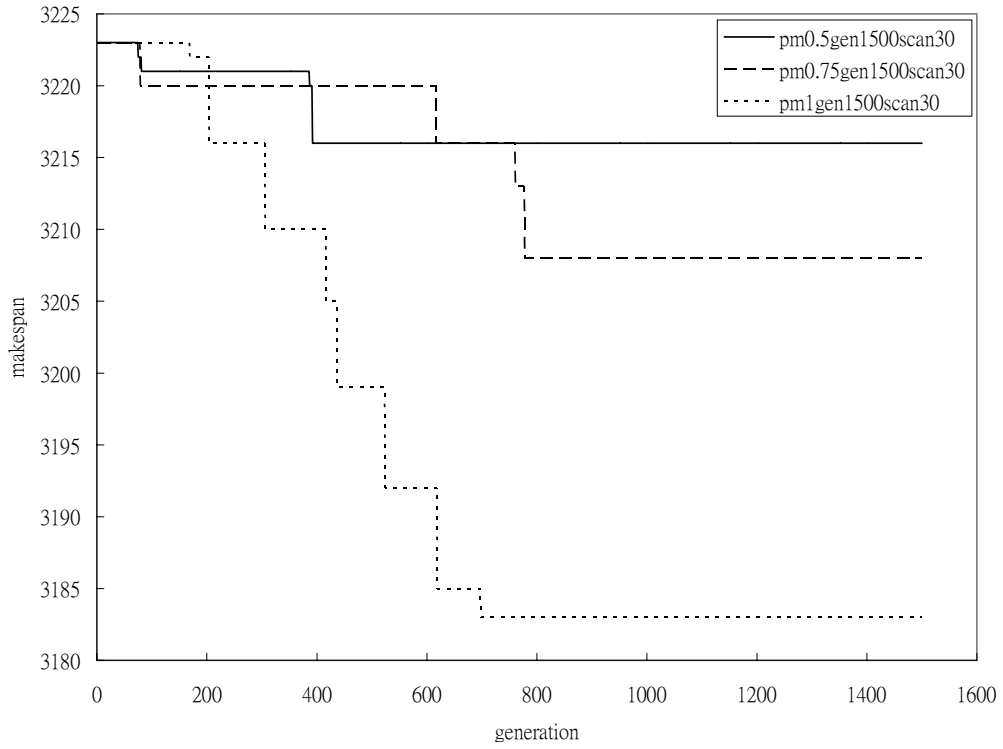


Figure 12. The further improvement of hybrid GA with initialization by improving heuristics in problem No. 16.

