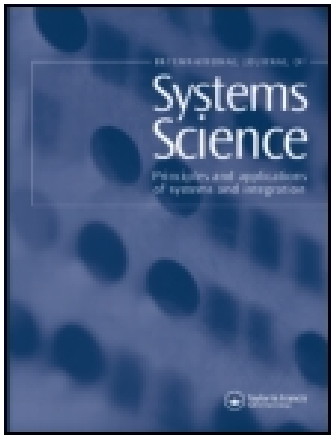

CrossMark

Click for updates

International Journal of Systems Science
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tsys20

Scheduling of multiple in-line steppers for semiconductor wafer fabs

Chie-Wun Chiou [a] & Muh-Cherng Wu [a]
[a] Department of Industrial Engineering and Management , National Chiao Tung University , Hsin-Chu 300 , Taiwan

Published online: 27 Sep 2012.

To cite this article: Chie-Wun Chiou & Muh-Cherng Wu (2014) Scheduling of multiple in-line steppers for semiconductor wafer fabs, International Journal of Systems Science, 45:3, 384-398, DOI: 10.1080/00207721.2012.724093

To link to this article: http://dx.doi.org/10.1080/00207721.2012.724093

International Journal of Systems Science, 2014
Vol. 45, No. 3, 384–398, http://dx.doi.org/10.1080/00207721.2012.724093

Scheduling of multiple in-line steppers for semiconductor wafer fabs

Chie-Wun Chiou and Muh-Cherng Wu*

Department of Industrial Engineering and Management, National Chiao Tung University, Hsin-Chu 300, Taiwan

(*Received 22 November 2010; final version received 1 August 2012*)

A few prior studies noticed that an in-line stepper (a bottleneck machine in a semiconductor fab) may have a capacity loss while operated in a low-yield scenario. To alleviate such a capacity loss, some meta-heuristic algorithms for scheduling a *single* in-line stepper were proposed. Yet, in practice, there are *multiple* in-line steppers to be scheduled in a fab. This article aims to enhance prior algorithms so as to deal with the scheduling for *multiple* in-line steppers. Compared to prior studies, this research has to additionally consider how to appropriately allocate jobs to various machines. We enhance prior algorithms by developing a *chromosome-decoding scheme* which can yield a job-allocation decision for any given chromosome (or job sequence). Seven enhanced versions of meta-heuristic algorithms (genetic algorithm, Tabu, GA–Tabu, simulated annealing, M-MMAX, PACO and particle swarm optimisation) were then proposed and tested. Numerical experiments indicate that the GA–Tabu method outperforms the others. In addition, the lower the process yield, the better is the performance of the GA–Tabu algorithm.

Keywords: scheduling; semiconductor; flow shop; port capacity constraints; genetic algorithm; meta-heuristic algorithms

1. Introduction

In semiconductor manufacturing, *in-line steppers* (or simply called *steppers*) are the most expensive machines, which may cost up to 40 million dollars per tool and usually become the bottleneck of a fab (semiconductor factory). Effective scheduling for steppers is very important because it could significantly affect the fab throughput, cycle time and on-time delivery.

Most prior studies on scheduling stepper are developed under a *high-yield* assumption (Chern and Liu 2003; Dabbas and Flowler 2003; Duwayri, Mollaghasemi, Nazzal, and Rabadi 2006; Sha, Hsu, Che, and Chen 2006; Wu, Huang, Chang, and Yang 2006; Morrison and Martin 2007; Wu and Chang 2007, 2008; Wu, Chiou, and Chen 2008; Wu, Jiang, and Chang 2008; Chen 2009). That is, they implicitly assumed the production yield is 100% and no wafer will be scrapped. Each wafer lot (a container for transporting wafers) is always a *full-lot* (typically carrying 25 wafers). Under this assumption, they took a stepper as a *single machine* and a *wafer lot* as a job for scheduling. This implies that a stepper will not be idle as long as it has wafer lots waiting to be processed.

Recently, a few studies (Chiou and Wu 2009; Wu and Chiou 2010) noticed that a stepper in a *low-yield* scenario may become *idle* even though it has many wafer lots waiting to be processed. They modelled the interior configuration of an in-line stepper as a *special-featured flow shop*, which comprises a series of chambers and each piece of wafer has to travel through the flow shop. They discovered that some chambers of the flow shop might become idle due to the inclusion of *small-lots* (i.e. carrying less than 25 wafers) and developed some scheduling algorithms to alleviate such chamber idleness in order to increase the throughput of steppers. These algorithms were developed in the context of scheduling a *single* in-line stepper.

However, in practice, there are *multiple* in-line steppers to be scheduled. This article enhances the prior scheduling algorithms to deal with the scheduling for *multiple* in-line steppers. Such a scheduling problem involves two decisions: (1) how to allocate jobs to each stepper and (2) how to sequence the allocated jobs for each stepper.

In this research, given N jobs to be allocated and scheduled on multiple steppers, a sequence of the N jobs is called a chromosome. We developed a novel *chromosome-decoding* scheme that can unveil the two decisions suggested by a given chromosome. Then, various enhanced versions of meta-heuristic algorithms (genetic algorithm (GA), Holland 1975; Tabu; GA–Tabu; simulated annealing (SA); M-MMAX; PACO; particle swarm optimisation (PSO)) were

*Corresponding author. Email: mcwu@mail.nctu.edu.tw

proposed and tested. Of the tested algorithms, numerical experiments indicate that the GA–Tabu outperforms the others in most cases.

The remainder of this article is organised as follows. Section 2 explains the research problem in more detail. Section 3 describes how to compute the performance of a scheduling solution. Section 4 presents the *chromosome-decoding* scheme. Section 5 describes the solution architecture of the GA–Tabu algorithm. Numerical experiments are reported in Section 6 and concluding remarks are in the last section.

2. Problem statement

The research problem is introduced by first describing the interior configuration, the exterior interfacing equipments and the transportation mechanisms of an in-line stepper. Then, we use an example to illustrate that such an in-line stepper may suffer a capacity loss in a low-yield scenario. Finally, the problem for scheduling multiple in-line steppers and its performance metric are presented.

The *interior configuration* of an in-line stepper comprises a sequence of *manufacturing stages*, each of which involves *one or more than one* functionally identical chambers (Quirk 2001). Each chamber processes *one* piece of wafer at a time. To undergo the operation at the stepper, a piece of wafer has to travel through all the manufacturing stages. Of these chambers, a particular type (called the aligner chamber) may need a setup. The aligner chamber is typically the bottleneck chamber and involves only one chamber at this manufacturing stage. Such a configuration can be seen as a *flow shop* if we consider each stage as a *workstation* and each chamber as a *machine* in a workstation (Yang 1999; Pinedo 2008). A simplified illustration of an in-line stepper is shown in Figure 1, where each manufacturing stage involves only one chamber.

The exterior of an in-line stepper is directly interfaced with a *dock area* which generally involves four *ports* (Figure 1). Each port serves as a

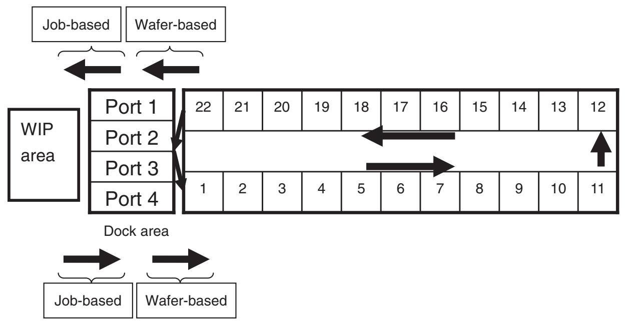

Figure 1. Configuration of a simplified in-line stepper.

one-job-buffer for the in-line stepper, which can accommodate only *one job* at a time. A job (or a wafer lot) is a container, which involves at most 25 pieces of wafers. Apart from the dock area, a large-sized stocker (also called the WIP area) is equipped to store the wafer jobs that are to be transported to the dock area.

The transportation mechanisms of an in-line stepper are explained below. A wafer job has to undergo a *round-trip travel* (Figure 1). A job first moves from the WIP area to the dock area, placed on a free port. Then, each piece of wafer will sequentially exit the job (a container), go through the in-line stepper and back to the job. Finally, the job is triggered to move back to the WIP area, while all its wafers complete operations. In summary, there is a *transportation incompatibility* in the round-trip travel. That is, the transportation unit between the WIP area and the dock area is a *job*, while that between the dock area and the chamber area is *a piece of wafer*.

Notice that the *buffer capacity* at the dock area is quite limited. Suppose the dock area involves only four ports. Then, the dock area can simultaneously accommodate at most four jobs. That is, we cannot move an additional job to the dock area if each of the four ports is currently occupied or not free.

Due to limited buffer capacity in the dock area, we may face a capacity loss of chambers in a low-yield scenario, as explained by the following example. Consider a simplified case where an in-line stepper comprises 22 stages and each stage involves only one chamber, four jobs (A, B, C and D) are on the dock area, and one job (E) is waiting in the WIP area. Job A contains 25 wafers and jobs B, C and D in total carry only 17 wafers. Suppose the processing sequence is $A \rightarrow B \rightarrow C \rightarrow D$. Following the sequence, the $(25+17)$ wafers of the four jobs will successively travel through the chambers and back to the ports. While the last wafer of job A just finishes its operations, job A must still stay on the port in order to get this wafer back. At this instant, the 17 unfinished wafers of jobs B, C and D will occupy the last 17 chambers of the stepper. The remaining five chambers then become idle because jobs A, B, C and D now occupy the dock area, and no more port is available for job E to access.

Yet, such a capacity loss in chambers may be alleviated if a different job sequence ($B \rightarrow C \rightarrow D \rightarrow A$) is applied. Suppose job B now contains eight wafers. While the last wafer (eighth one) of job B finishes its travel, we still have 34 unfinished wafers. Of these unfinished wafers, 22 ones reside in the chamber area and 12 ones stay in the ports. This implies that no chamber will be idle while the port originally for job B becomes free to accommodate job E. Thus, the capacity loss of an in-line stepper may be alleviated by applying an appropriate job sequence.

The research problem is concerned with the scheduling of multiple in-line steppers in a low-yield scenario. Such a scheduling problem involves two decisions: (1) how to allocate jobs to each stepper and (2) how to sequence the allocated jobs for each stepper. That is, given N jobs to be scheduled for m in-line steppers. The N jobs have to be categorised into m groups, and the jobs of each group have to be sequenced. The performance metric of the scheduling problem is defined as $C^* = \max\{C_1, \ldots, C_m\}$, where C_i is the makespan of ith stepper. That is, C^* is the makespan required to complete the N jobs, and N/C^* could be used to denote the total throughput rate of the m steppers. In the scheduling, we aim to maximise total throughput rate (N/C^*), which also implies the minimisation of C^*.

3. Makespan evaluation for job sequences

Given a job sequence to be processed by the ith stepper, this section describes how to compute C_i (the makespan), adapted from Ruiz and Maroto (2006). The makespan evaluation procedure adopts an *emulation-based* approach. We virtually sent each wafer in order (following the job sequence) into the in-line stepper and look for an available chamber that can finish the job at the earliest time. The completion time of each wafer at each stage is progressively recorded, which ultimately yield the makespan.

To undergo an operation at an aligner chamber, a mask (an auxiliary device) is needed. A particular mask denotes a particular operational recipe. Different jobs may require different operational recipes. If so, we need a setup time to change masks at the aligner chamber; otherwise no setup is needed.

Notation

j	index of job
k	index of wafer
i	index of stage
l	index of chamber
a	index of the aligner chamber which requires setup
ρ	total number of ports in the dock
n	total number of jobs to be processed by the in-line stepper
M	total number of stages in the in-line stepper
m_i	total number of chambers at stage i
p_{iljk}	processing time required by chamber l at stage i to process wafer k in job j
π	a job sequence for the n jobs, $\pi = [\pi(1), \ldots \pi(n)]$
$w(\pi(j))$	the job in the jth position of sequence π
$\pi(j)$	total number of wafers in job j
t_u	transportation time for uploading a job to the dock area
t_d	transportation time for downloading a job from the dock area
$S_{i,l,\pi(j),k}$	setup time required by chamber l in stage i to process wafer k in job $\pi(j)$

$$\begin{aligned} &\text{if } i \neq a \text{ or } k \neq 1, \quad \text{then } S_{i,l},\pi(j),k = 0,\\ &\text{otherwise,} \qquad\qquad S_{i,l},\pi(j),k = \delta_{\pi(j),\pi(j-1)} \end{aligned}$$

$\delta_{\pi(j),\pi(j-1)}$	setup time required for the aligner chamber to switch production from job $\pi(j-1)$ to job $\pi(j)$; $\delta_{\pi(j),\pi(j-1)} = s_0$ if $\pi(j-1)$ and $\pi(j)$ use different masks, and $\delta_{\pi(j),\pi(j-1)} = 0$, otherwise
$A_{i,l,t}$	the time epoch when chamber l in stage i just turns to be available; that is, while the chamber (i, l) is free at t, $A_{i,l,t}$ is the last *wafer-completion-epoch* before t; while the chamber (i, l) is in operation at t, $A_{i,l,t}$ is the *first wafer-completion-time* after t.
$C_{i,\pi(j),k}$	the completion time of wafer k in job $\pi(j)$ at stage i
$C(\pi)$	the makespan of job sequence π

The makespan evaluation procedure is governed by the following equations:

$$\begin{aligned} &C_{i,\pi(j),k}\\ &\quad = \min_{1\le l\le m_i} \left\{ \max\{A_{i,l,t} + S_{i,l,\pi(j),k}, C_{i-1,\pi(j),k}\} + p_{i,l,\pi(j),k} \right\}, \end{aligned}$$

$$\text{where } t = C_{i-1,\pi(j),k} \quad \text{for } 1 \le i \le M, \tag{1}$$

$$C_{M+1,\pi(j),w(\pi(j))} = C_{M,\pi(j),w(\pi(j))} + t_d, \tag{2}$$

$$C_{M+1,\pi(j),w(\pi(j))} + t_u = C_{0,\pi(j+\rho),1} \quad \text{for } 1 \le j \le n-\rho, \tag{3}$$

$$C(\pi) = C_{M+1,\pi(n),w(\pi(n))}. \tag{4}$$

Equation (1) expresses the completion time of a particular wafer at each stage i. The term $A_{i,l,t} + S_{i,l,\pi(j),k}$ denotes the time epoch when chamber l at stage i is ready for processing wafer k in job $\pi(j)$, and the term $C_{i-1,\pi(j),k}$ denotes the time epoch when the wafer is available to be processed at the chamber.

Equation (2) describes the completion time of job $\pi(j)$ at stage $M+1$ (i.e. the dock area). Equation (3) expresses the job arrival/departure relationships for the dock. The equation indicates that only when job $\pi(j)$ in the dock has been moved away, can job $\pi(j+\rho)$ in the WIP buffer be transported to the dock (i.e. stage 0).

Equation (4) computes the makespan $C(\pi)$, the completion time of the last wafer in the last job.

4. Chromosome representation and decoding

As stated in Section 1, the scheduling problem involves two decisions: (1) *allocation decision* – how to allocate jobs to each stepper and (2) *sequencing decision* – how to sequence the allocated jobs for each stepper. That is, given N jobs to be scheduled on m steppers, we first need to categorise the N jobs into m groups, each of which is processed by a particular stepper. Then, we need to determine the job sequence for each stepper.

Let a particular sequence of the N jobs be called a *chromosome*. In this research, we develop a *chromosome-decoding* scheme. By the decoding scheme, a chromosome can be interpreted as a particular scheduling solution, which involves two decisions – one is the allocation decision and the other is the sequencing decision.

To introduce the *chromosome-decoding* scheme, some notation is described below. Denote a *chromosome* by $\pi = [\pi(1), \ldots \pi(N)]$, where $\pi(j)$, called a *gene*, represents the job in the jth position of sequence π. For job $\pi(j)$, represent the number of wafers in the job by $w_{\pi(j)}$ and its processing time for a piece of wafer at the aligner chamber (the bottleneck chamber) by $p_{\pi(j)}$. Then, denote the total processing time at the aligner chamber for job $\pi(j)$ by $t_{\pi(j)} = w_{\pi(j)} \cdot p_{\pi(j)}$. The procedure for interpreting the job allocation decision from a given chromosome is described below.

Procedure *Job_Allocation*

```
Step 1: Compute the threshold for forming a job group
    $T = \sum_{j=1}^{N} t_{\pi(j)}$;   /*total processing time of the N jobs*/
    $h = T/m$;   /*processing time threshold for forming a job group*/
Step 2: Form the job groups
    $k = 1$, /*index of job group or stepper*/
    For $i = 1$ to $N$
        $T_i = \sum_{j=1}^{i} t_{\pi(j)}$;   /*compute total load of the first i jobs*/
        If $(T_i > k \cdot h)$ then   /*criterion for forming job groups*/
            $C(k) = i$;   /*form a new job group*/
            $k = k + 1$;   /*update the indexing of job group*/
        Endif
        If $(k = m)$ then
            go to Step 3   /*check if job group formation finished*/
        Endif
    Endfor
Step 3: Output job allocation results
        Output $C(k)$, $1 \le k \le m - 1$
```

Given the job allocation decision $C(k)$, the procedure for determining the job sequence decision for each stepper is relatively easy. The job sequence for stepper k $(1 \le k \le m)$ is $\pi_k = [\pi(s), \ldots \pi(e)]$, where $s = C(k-1)+1$ and $e = C(k)$, in which we denote $C(0) = 0$ and $C(m) = \pi(N)$.

The chromosome-decoding scheme is illustrated by a three-stepper example as shown in Figure 2. In the figure, there are nine jobs to be scheduled on three steppers. The total processing time is $T = 1.6$ h and the threshold is $h = 0.53$ h. The set of jobs allocated to stepper 1 and their job sequence are $\pi_1 = \{J_2, J_5, J_7\}$, and those for the other two steppers are $\pi_2 = \{J_4, J_1\}$ and $\pi_3 = \{J_6, J_3, J_9, J_8\}$.

5. GA–Tabu algorithm

To solve the scheduling problem, we developed seven meta-heuristic algorithms (GA, Tabu, GA–Tabu, SA, MMAX, PACO and PSO). These algorithms adopt the algorithmic architectures published in prior studies as referenced below, but are distinguished in embedding the novel *chromosome decoding* scheme we proposed. Of these seven enhanced algorithms, the GA–Tabu performs the best and is presented here.

5.1. *Chromosome fitness*

As stated in Section 4, given a chromosome $\pi = [\pi(1), \ldots \pi(N)]$, we can use the chromosome decoding scheme to *extract* its two decisions – jobs allocation among steppers and job sequencing for each stepper. Then, given a job sequence $\pi_k = [\pi(s), \ldots \pi(e)]$ for *each* stepper k, $1 \le k \le m$, we can compute its makespan C_k by the procedure in Section 3. In turn, the scheduling performance (also called *fitness*) of the chromosome $\pi = [\pi(1), \ldots \pi(N)]$ is $C^* = \max\{C_1, \ldots, C_m\}$, which is also denoted by $C^*(\pi)$ hereafter.

5.2. *Algorithmic procedures*

The GA–Tabu algorithm is composed of three procedures. The main one is called procedure *GA–Tabu* which calls two sub-procedures $GA(t)$ and $Tabu(\pi_{\text{in}}, \pi_{\text{out}})$. There are two sets of chromosomes. One is called the GA-pool $P(t)$, which include N chromosomes and iteratively evolve by procedure $GA(t)$. The other set is called *Seed_Set*, which involves

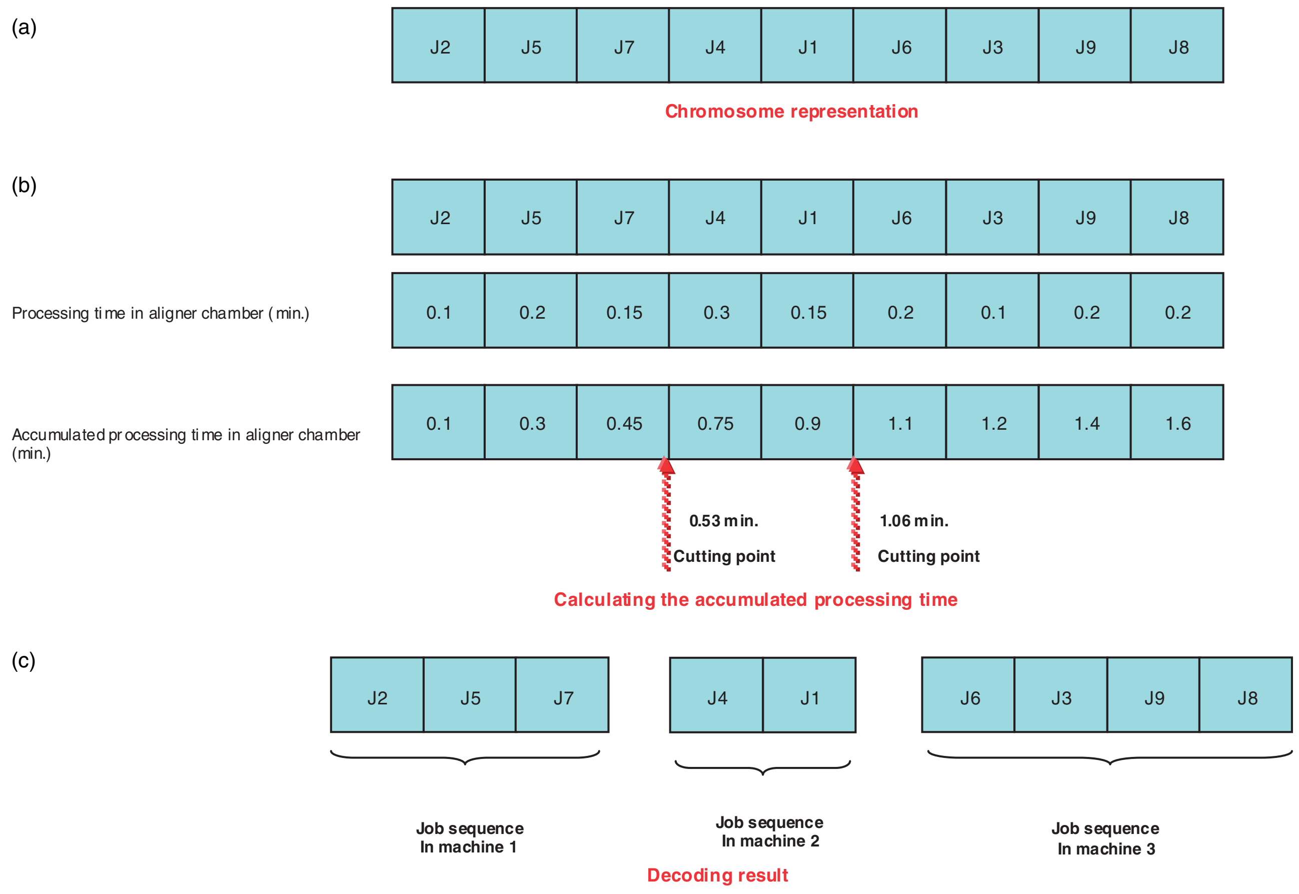
(a)
J2 J5 J7 J4 J1 J6 J3 J9 J8
Chromosome representation
(b)
J2 J5 J7 J4 J1 J6 J3 J9 J8
Processing time in aligner chamber (min.)
0.1 0.2 0.15 0.3 0.15 0.2 0.1 0.2 0.2
Accumulated processing time in aligner chamber (min.)
0.1 0.3 0.45 0.75 0.9 1.1 1.2 1.4 1.6
0.53 min.
Cutting point
1.06 min.
Cutting point
Calculating the accumulated processing time
(c)
J2 J5 J7
J4 J1
J6 J3 J9 J8
Job sequence In machine 1
Job sequence In machine 2
Job sequence In machine 3
Decoding result

Figure 2. GA–Tabu chromosome for three in-line steppers.

only one chromosome (called seed) and iteratively evolve by procedure *GA–Tabu*. The procedure $GA(t)$ is designed to evolve from $P(t)$ to $P(t+1)$ to possibly include better chromosomes and identify its best four ones. The procedure $Tabu(\pi_{\text{in}}, \pi_{\text{out}})$ is intended to search the neighbourhood of a given chromosome π_{in} to find the best chromosome (π_{out}) in the searched neighbourhood. Notice that π_{in} is selected either from the *GA-pool* or from the *Seed_Set*.

In procedure $GA(t)$, we use four types of crossover operators and three types of mutation operators to create new chromosomes. A crossover operator is designed to create a new pair from an existing pair, while a mutation operator is to create a new one from an existing one. The four types of crossover operators are: *C1* (one point crossover) by Reeves (1995), *LOX* (linear order crossover) by Croce, Tadei, and Volta (1995), *PMX* (partially mapped crossover) by Goldberg (1989) and *NABEL* operator by Bac and Perov (1993). The three types of mutation operators are *Swap, Inverse* and *Insert* (Wang and Zheng 2003; Nearchou 2004).

In procedure $Tabu(\pi_{\text{in}}, \pi_{\text{out}})$, a pairwise interchange of two genes (jobs) is called a *tabu_move*. For example, given a n-gene chromosome $\pi_1 = [J_2, J_5, J_1, J_4, J_3 \ldots]$. By interchanging the two genes J_3 and J_5, we can create a new chromosome $\pi_2 = [J_2, J_3, J_1, J_4, J_5, \ldots]$. The tabu_move for causing such an interchange can be denoted by $move(J_5, J_3)$ or $move(J_3, J_5)$. To facilitate the following presentation of procedure $Tabu(\pi_{\text{in}}, \pi_{\text{out}})$, we represent a tabu_move by $move(\pi_{\text{in}} \to \pi)$, which denotes an interchange of two particular jobs that transform π_{in} into π.

Accordingly, the total number of *tabu_moves* for a chromosome $\pi = [\pi(1), \ldots \pi(n)]$ is $n(n-1)/2$. Let the set of all these tabu_moves be represented by *Move_Set*. By applying each tabu_move in the *Move_Set* to the chromosome π, we can create $n(n-1)/2$ new chromosomes. The set of these newly created chromosomes are called the neighbourhood of π, which is denoted by Neighbour(π).

Notation

GA–Tabu	the main procedure
π^{best}	the current best solution ever found by procedure *GA–Tabu*
$GA(t)$	a sub-procedure designed to evolve $P(t)$

$P(t)$	a set of N chromosomes, called the *GA-pool*
$\pi^{\text{best}}_{GA,t}$	the best chromosome in $P(t)$
$\pi^{2-\text{best}}_{GA,t}$	the second best chromosome in $P(t)$
$\pi^{3-\text{best}}_{GA,t}$	the third best chromosome in $P(t)$
$\pi^{4-\text{best}}_{GA,t}$	the fourth best chromosome in $P(t)$
Tabu $(\pi_{\text{in}}, \pi_{\text{out}})$	a procedure designed to search the neighbourhood of π_{in}, where π_{out} is the best chromosome found in the searched region.
Tabu_list	a limited set of tabu_moves, where a tabu_move represents an interchange of two particular jobs.
Seed_Set	a set of one chromosome, which iteratively evolve by procedure *GA–Tabu*
π^{seed}	the present chromosome (called seed) in the *Seed_Set*

Procedure GA–Tabu

```
Initialisation: Randomly select a chromosome as $\pi^{\text{best}}$

  For each iteration $t$ $(0 \le t \le T_f)$

Step 1: Call $GA(t)$ to find $\pi^{\text{best}}_{GA,t+1}$, $\pi^{2-\text{best}}_{GA,t+1}$, $\pi^{3-\text{best}}_{GA,t+1}$, $\pi^{4-\text{best}}_{GA,t+1}$ in $P(t+1)$
Step 2: If $\pi^{\text{best}}_{GA,t+1}$ is better than $\pi^{\text{best}}$
    • Update $\pi^{\text{best}}$ by $\pi^{\text{best}}_{GA,t+1}$ (i.e. $\pi^{\text{best}} \leftarrow \pi^{\text{best}}_{GA,t+1}$)
    • Call Tabu $(\pi^{\text{best}}, \pi_{\text{out}})$ to search the neighbourhood of $\pi^{\text{best}}$ for possibly improving $\pi^{\text{best}}$ (i.e. update $\pi^{\text{best}}$ by $\pi_{\text{out}}$ if $\pi_{\text{out}}$ is better)
Step 3: If $\pi^{\text{best}}_{GA,t+1}$ keeps worse than $\pi^{\text{best}}$ for exact $k$ $(k < 45)$ iterations
    • While $k = 20$ /* the age of $\pi^{\text{best}}$ is 20 iterations*/
      Call Tabu $(\pi^{2-\text{best}}_{GA,t+1}, \pi_{\text{out}})$ for possibly improving $\pi^{\text{best}}$
    • While $k = 30$ /*the age of $\pi^{\text{best}}$ is 30 iterations*/
      Call Tabu $(\pi^{3-\text{best}}_{GA,t+1}, \pi_{\text{out}})$ for possibly improving $\pi^{\text{best}}$
    • While $k = 40$ /* the age of $\pi^{\text{best}}$ is 40 iterations*/
      Call Tabu $(\pi^{4-\text{best}}_{GA,t+1}, \pi_{\text{out}})$ for possibly improving
Step 4: If $\pi^{\text{best}}_{GA,t+1}$ keeps worse than $\pi^{\text{best}}$ for $k = 40 + 5n$ $(n = 1, 2, \ldots)$ iterations
      /* If $\pi^{\text{best}}$ cannot be improved by $P(t+1)$ for over 40 iterations*/
      /* Try to improve $\pi^{\text{best}}$ by using $\pi^{\text{seed}}$, the chromosome in Seed_Set*/
    • Call Tabu $(\pi^{\text{seed}}, \pi_{\text{out}})$ for possibly improving $\pi^{\text{best}}$
    • Update the chromosome in Seed_Set (i.e. $\pi^{\text{seed}} \leftarrow \pi_{\text{out}}$)
Step 5: Put $\pi^{\text{best}}$ in $P(t+1)$;

Next iteration until $t > T_f$
```

Procedure *GA*(*t*)

```
Step 1: If $t = 0$, randomly create $N$ chromosomes to form the initial GA-pool $P(0)$.
        If $t \ne 0$, input the GA-pool $P(t)$.
Step 2: Use crossover and mutation operators to create $N(P_{cr}+P_{mu})$ new chromosomes, where $0 \le P_{cr}, P_{mu} \le 1$. Place the new chromosomes in a set $S$.
Step 3: Use the resolute wheel screen method (Michalewicz 1996) to select $N$ chromosomes out of the set $S \cup P(t)$. Place the selected $N$ chromosomes in the GA-pool $P(t+1)$
Step 4: Output the best four chromosomes in $P(t+1)$: $\pi^{\text{best}}_{GA,t+1}$, $\pi^{2-\text{best}}_{GA,t+1}$, $\pi^{3-\text{best}}_{GA,t+1}$, $\pi^{4-\text{best}}_{GA,t+1}$.
```

Procedure Tabu($\pi_{\text{in}}, \pi_{\text{out}}$)

```
Step 0: Initialisation $\pi_{\text{out}} = \pi_{\text{best}}$
Step 1: Create a set of new chromosomes, Neighbour$(\pi_{\text{in}})$
Step 2: Try to improve $\pi_{\text{out}}$
    • Identify the best $q$ chromosome $\pi^1, \ldots, .\pi^q$ from Neighbour$(\pi_{\text{in}})$
    • If $\pi^1$ is better than $\pi^{\text{best}}$, then $\pi_{\text{out}} = \pi^1$;
Step 3: Update tabu_list /*Try to add a new tabu_move to the tabu_list*/
  For $i = 1, q$
    If move$(\pi_{\text{in}} \to \pi^i) \in$ tabu_list, then go to Next $i$
    If move$(\pi_{\text{in}} \to \pi^i) \notin$ tabu_list,
      Then
        • Update the tenure of each tabu_move in the tabu_list
          /* Each tabu_move's tenure starts at 1 and is added by 1 while a new tabu_move is found*/
        • Remove the tabu_move with longest tenure from the tabu_list;
        • Put move$(\pi_{\text{in}} \to \pi^i)$ in the tabu_list;
        • If $(\pi_{\text{in}} = \pi^{\text{seed}})$, set $\pi^{\text{seed}} = \pi^i$
        • Go to Step 4
  Next $i$
Step 4: Return
```

6. Experiments and discussion

Numerical experiments and the uniqueness of this research are to be discussed in this section. In numerical experiments, we compare seven meta-heuristic algorithms in solving the multiple-steppers scheduling problem. These algorithms are chosen because their algorithmic flows have been widely applied to solving scheduling problem. Referring to some prior studies, we adapted these algorithmic flows by embedding the novel chromosome interpretation scheme.

These prior related studies include GA–Tabu by Chiou and Wu (2009), SA by Osman and Potts (1989), GA by Wu and Chiou (2010), Tabu search by Widmer and Hertz (1989), PSO by Liao, Tseng and Luarn (2007) and two ant colony algorithms (ACOs) by Rajendran and Ziegler (2004) – respectively, called MMAX and PACO.

Personal computers equipped with PENTIUM Dual-Core 2.8 GHz CPU and 1 Gb memory are used to run the programs, coded in Visual C++. The parameters of the six meta-heuristic algorithms are designed by referring to prior studies accordingly. In the three algorithms (GA–Tabu, GA and Tabu), we set $N = 100$, $P_{cr} = 0.8$, $P_{mu} = 0.2$, $q = 7$, $K = 3000$, $T = 100{,}000$ (Widmer and Hertz 1989; Chiou and Wu 2009). In the SA, we set $Temperature = 500$ (Osman and Potts 1989); and we set $\rho = 0.75$ (Rajendran and Ziegler 2004) in the ACO where $(1-\rho)$ is called the evaporation rate. In the PSO, we set $V_{\max} = 4$, $V_{\min} = -4$, $\varpi = 0.8$, $c_1 = 2$, $c_2 = 2$, $T = 100$ (Liao et al. 2007), where $V_{\max}$ denotes the upper limit of the velocity, $V_{\min}$ denotes the lowest limit of the velocity, c_1 is the cognition learning factor, c_2 is the social learning factor and ϖ is the inertia weight.

6.1. *Experiment design*

In the experiments, the configuration and operation of the in-line steppers are so defined. Each in-line stepper has four ports, 14 stages and 21 chambers. Of the 14 stages, two stages model the interfaces among the WIP buffer, the dock area and the stepper; and the other 12 stages model the interior chambers of the stepper (Table 1). Note that one stage may include one or more chambers. The operation time at each stage i of each in-line stepper is a uniform distribution $[a_i, b_i]$. At each stepper, a mask setup is always needed for the aligner chamber while it turns to process a new job's wafer, and the mask change time is a constant (1.0 min). The process yields are modelled by *truncated binomial distributions*, which denote that the job size is first generated by a *binomial distribution*, and then those jobs that carry no wafer are 'truncated' (removing them from the fab).

We use (M, N, Y) to represent a test case, where M represents the number of in-line steppers, N represents the number of jobs and Y represents the average yield. We design 100 test cases, in which M has two options (with two or three steppers) and N has five options ranging from 20 to 100 jobs and Y includes 10 options ranging from 15% to 90%. In practice, the complexity of a typical multiple-stepper scheduling problem is $M = 2$ and $N = 60$.

In each test case, we run 15 replicates and the average *makespan* of the 15 replicates is taken as the performance measure of the tested algorithm. The *average makespan* of each algorithm x is designated as C_x. For example, $C_{GA-Tabu}$ represents the average makespan of the GA–Tabu. Likewise, the average CPU time used in each algorithm x is defined as t_x; for example, that used in the GA–Tabu is represented by $t_{GA-Tabu}$.

Pilot experiments indicate that the GA–Tabu essentially outperforms the other six algorithms. To compare the solution quality of the seven algorithms, we define a *performance gap* as follows: $\gamma_x = (C_x - C_{GA\_Tabu})/C_{GA\_Tabu}$. A positive γ_x indicates that the GA–Tabu outperforms the x algorithm, while a negative γ_x denotes that the GA–Tabu is inferior to the x algorithm.

6.2. *Experiment results*

Tables 2–11 show the experiment results of γ_x and t_x. The GA–Tabu outperforms the other six algorithms in terms of γ_x. Of the 100 test cases, γ_x ranges from 0% to 22% but the average of γ_x is only 1.19%. The lower is Y (average process yield), the higher is the average of γ_x (Figure 3). While Y decreases to 15%, the average of γ_x even reaches up to 8.06%. This is because the number of small lots in a high-yield scenario is fewer which in turn results in less capacity loss. Therefore, the GA–Tabu outperforms the other algorithms substantially in low-yield scenarios and slightly in high-yield scenarios.

Of the 100 test cases, the average of γ_x for each algorithm is shown in Figure 4, which indicates that the Tabu algorithm is the second best – the average of γ_{Tabu} is only 0.16%. However, the value of γ_{Tabu} may reach up to 5.31% in a low-yield scenario (Table 3). The average of t_x for each algorithm is shown in Figure 5, which indicates that the Tabu algorithm is computationally faster than the GA–Tabu. The maximum value of $t_{GA-Tabu}$ is 4379 s (about 1.2 h). In practice, the scheduling decision is made on every working shift (12 h); solving such a scheduling problem 1.5 h before the shift is acceptable to practitioners.

Table 1. Process times of in-line stepper chambers.

Process sequence	WIP buffers to dock area	Dock area to track	HMDS	Cooling	Coater	Soft-bake	Cooling	Aligner	Wafer edge exposure	PEB	Cooling	Develop	Hard bake	High cooling
Chamber number	1	1	2	2	2	2	2	1	1	2	2	2	2	1
Process time (min)	2.5	0.1	1.2	1.2	1.2	[1.2, 2.8]	1	[0.75, 1.65]	1	[1.2, 2.8]	1	[1.2, 2.8]	[1.2, 2.8]	0.5

Table 2. Performance comparisons of algorithms in scenarios with $(M, N) = (2, 20)$.

	20													
Jobs	GA_Tabu		Tabu		GA		PACO		MMAX		SA		PSO	
Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)	r_{PACO} (%)	t_{PACO} (sec)	r_{MMAX} (%)	t_{MMAX} (sec)	r_{SA} (%)	t_{SA} (sec)	r_{PSO} (%)	t_{PSO} (sec)
90	269.7	119	0.00	79.0	0.09	244	0.59	21	0.36	29	1.12	6	1.94	2
80	254.8	109	0.11	79.0	0.14	296	0.28	19	0.18	28	1.04	7	1.91	1
70	216.0	104	0.05	78.0	0.09	253	0.41	16	0.23	23	1.43	5	2.01	1
60	186.3	103	0.08	77.9	0.23	217	0.37	14	0.30	20	1.34	5	2.15	1
50	168.9	96	0.17	77.1	0.26	196	0.39	13	0.36	18	1.61	4	2.66	1
40	130.5	63	0.14	77.0	0.36	157	0.50	10	0.49	13	1.94	4	3.21	1
30	106.5	65	0.08	76.0	0.29	160	0.81	8	0.39	10	1.98	3	3.35	1
25	81.1	46	0.15	75.1	0.71	156	0.86	6	0.79	8	3.16	2	5.61	1
20	70.2	42	0.59	75.0	0.89	129	2.26	5	1.72	6	5.24	2	7.85	1
15	62.5	37	0.25	75.0	1.50	118	2.47	4	2.31	5	7.03	2	10.63	1

Table 3. Performance comparisons of algorithms in scenarios with $(M, N) = (2, 40)$.

	40													
Jobs	GA_Tabu		Tabu		GA		PACO		MMAX		SA		PSO	
Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)	r_{PACO} (%)	t_{PACO} (sec)	r_{MMAX} (%)	t_{MMAX} (sec)	r_{SA} (%)	t_{SA} (sec)	r_{PSO} (%)	t_{PSO} (sec)
90	564.6	381	0.05	160.0	0.02	722	0.10	185	0.10	261	0.97	30	1.37	7
80	488.1	333	0.07	155.0	0.04	635	0.20	160	0.20	226	1.06	31	1.59	7
70	438.5	294	0.08	151.1	0.18	674	0.29	143	0.29	203	1.26	30	1.79	6
60	378.3	276	0.03	146.0	0.00	553	0.16	122	0.16	173	1.33	28	2.00	6
50	331.9	261	0.08	143.0	0.00	488	0.16	106	0.16	150	1.42	26	2.11	6
40	254.4	182	0.00	138.0	0.11	359	0.20	81	0.20	116	1.89	24	2.87	6
30	200.0	150	0.07	133.2	0.10	334	0.45	63	0.45	89	2.29	22	3.62	5
25	172.2	150	0.04	130.2	0.29	419	0.52	54	0.52	76	3.39	21	4.93	5
20	129.9	128	5.31	129.3	2.20	386	1.90	48	1.90	69	15.55	20	15.77	5
15	118.8	104	2.71	124.3	2.80	649	3.66	34	3.66	48	22.91	19	15.34	5

Table 4. Performance comparisons of algorithms in scenarios with $(M, N) = (2, 60)$.

	60													
Jobs	GA_Tabu		Tabu		GA		PACO		MMAX		SA		PSO	
Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)	r_{PACO} (%)	t_{PACO} (sec)	r_{MMAX} (%)	t_{MMAX} (sec)	r_{SA} (%)	t_{SA} (sec)	r_{PSO} (%)	t_{PSO} (sec)
90	840.4	961	0.02	526.7	0.04	1670	0.05	618	0.14	876	0.80	25	1.17	17
80	738.7	838	0.01	510.6	0.02	1245	0.06	542	0.28	773	1.01	23	1.51	17
70	649.8	792	0.09	495.0	0.03	1549	0.13	475	0.16	679	1.01	22	1.45	16
60	585.9	694	0.01	483.5	0.07	1091	0.15	427	0.29	612	1.22	19	1.74	16
50	473.4	517	0.01	465.8	0.13	980	0.18	339	0.28	487	1.01	18	1.88	15
40	391.3	433	0.00	452.3	0.19	771	0.30	278	0.35	401	1.80	16	2.40	15
30	278.4	324	0.13	430.7	0.45	726	0.28	197	0.43	282	1.98	14	3.58	15
25	234.5	284	0.07	421.3	1.11	858	0.76	166	0.77	240	3.81	13	5.98	15
20	207.1	281	0.32	417.3	3.26	1384	2.18	149	2.06	216	7.78	11	9.50	14
15	170.3	212	1.66	410.5	9.02	1595	5.00	112	6.26	162	21.00	10	16.78	14

Table 5. Performance comparisons of algorithms in scenarios with $(M, N) = (2, 80)$.

	80													
Jobs	GA_Tabu		Tabu		GA		PACO		MMAX		SA		PSO	
Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)	r_{PACO} (%)	t_{PACO} (sec)	r_{MMAX} (%)	t_{MMAX} (sec)	r_{SA} (%)	t_{SA} (sec)	r_{PSO} (%)	t_{PSO} (sec)
90	1123.3	1851	0.01	1209.9	0.03	2502	0.12	1474	0.13	2084	0.79	49	1.05	34
80	1004.2	1731	0.06	1174.1	0.09	2722	0.27	1309	0.37	1857	0.80	47	1.18	33
70	891.7	1659	0.00	1138.5	0.06	2053	0.16	1158	0.19	1648	0.86	43	1.30	33
60	760.3	1352	0.00	1098.2	0.06	1767	0.21	978	0.30	1393	1.58	39	1.47	32
50	654.7	1130	0.02	1074.0	0.05	1726	0.20	844	0.21	1205	1.26	36	1.75	32
40	515.0	864	0.05	1033.7	0.25	1109	0.35	660	0.43	954	2.66	32	2.16	31
30	412.5	708	0.00	998.1	0.22	1262	0.59	521	0.20	744	1.77	28	2.62	31
25	334.7	601	0.01	966.9	0.89	1315	1.12	426	0.55	610	4.69	26	4.11	30
20	292.4	529	0.07	954.1	2.69	1997	2.39	374	1.87	543	5.85	24	7.39	30
15	228.9	392	2.68	931.3	6.91	2126	7.80	253	4.02	366	8.77	21	16.64	30

Table 6. Performance comparisons of algorithms in scenarios with $(M, N) = (2, 100)$.

	100													
Jobs	GA_Tabu		Tabu		GA		PACO		MMAX		SA		PSO	
Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)	r_{PACO} (%)	Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)
90	1398.8	3402	0.02	2328	0.06	4002	0.15	2851	0.16	4053	0.76	31	1.03	58
80	1256.4	3348	0.06	2261	0.09	3250	0.37	2546	0.37	3636	0.78	28	1.09	58
70	1120.1	2734	0.03	2193	0.03	3459	0.15	2253	0.18	3229	0.78	25	0.97	57
60	959.0	2294	0.04	2117	0.10	2389	0.25	1920	0.30	2769	1.04	22	1.38	57
50	785.2	1868	0.06	2041	0.08	2454	0.23	1568	0.26	2257	1.11	18	1.63	56
40	658.1	1711	0.03	1988	0.19	2183	0.28	1308	0.37	1886	1.44	16	2.14	55
30	501.4	1231	0.00	1900	0.45	1780	0.83	984	0.43	1405	2.76	12	2.69	55
25	451.2	1153	0.51	1870	0.66	1760	0.91	882	0.29	1258	2.60	11	2.20	55
20	364.2	961	0.47	1825	3.30	2952	3.03	728	2.96	1055	7.52	9	7.73	54
15	303.1	681	0.09	1783	5.75	3151	6.69	511	3.49	740	11.40	6	11.95	54

Table 7. Performance comparisons of algorithms in scenarios with $(M, N) = (3, 20)$.

	20													
Jobs	GA_Tabu		Tabu		GA		PACO		MMAX		SA		PSO	
Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)	r_{PACO} (%)	Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)
90	197.9	4379	0.11	6.0	0.09	5876	0.41	21	0.29	31	1.27	30	1.37	2
80	186.7	3970	0.19	5.0	0.13	4726	0.59	19	0.31	29	1.47	27	1.61	2
70	159.7	3484	0.14	4.0	0.22	4348	0.50	16	0.25	25	1.53	24	1.77	1
60	140.9	3375	0.12	4.0	0.39	4978	0.69	14	0.58	21	1.60	21	1.87	1
50	126.7	2450	0.11	3.0	0.57	3996	0.76	13	0.61	19	1.69	17	2.07	1
40	99.8	2046	0.10	2.0	0.44	4059	0.55	10	0.52	14	1.81	14	2.29	1
30	82.2	1540	0.06	2.0	0.55	2632	0.60	7	0.51	11	1.83	11	2.68	1
25	63.7	1377	0.13	1.0	0.81	3063	1.08	6	0.85	8	2.56	10	3.88	1
20	55.9	1135	0.39	1.0	0.89	2209	1.18	5	1.11	7	3.43	9	4.99	1
15	49.2	896	0.57	1.0	1.53	2238	1.99	4	1.53	6	5.10	7	8.51	1

Table 8. Performance comparisons of algorithms in scenarios with $(M, N) = (3, 40)$.

	40													
Jobs	GA_Tabu		Tabu		GA		PACO		MMAX		SA		PSO	
Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)	r_{PACO} (%)	Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)
90	394.0	416	0.10	54.0	0.15	271	0.07	1176	0.13	178	1.38	12	1.59	7
80	342.1	369	0.07	47.0	0.11	234	0.13	1042	0.11	155	1.60	11	1.83	7
70	308.7	330	0.07	42.0	0.19	210	0.25	1021	0.31	139	1.78	10	1.99	6
60	266.0	287	0.14	36.0	0.35	180	0.19	843	0.28	119	1.80	9	2.31	6
50	234.2	266	0.08	31.0	0.28	156	0.24	879	0.18	104	1.77	8	2.53	6
40	181.1	208	0.07	24.0	0.45	121	0.47	624	0.37	81	2.45	6	3.14	6
30	143.2	175	0.23	19.0	0.53	94	0.52	728	0.38	63	2.92	5	3.92	5
25	124.0	150	0.27	16.0	0.52	80	0.67	670	0.70	54	3.98	4	5.27	5
20	94.5	170	3.43	12.0	2.82	61	0.66	720	2.67	41	12.11	4	15.36	5
15	85.4	118	0.75	10.0	2.47	50	2.02	567	3.27	34	10.90	3	15.91	5

Table 9. Performance comparisons of algorithms in scenarios with $(M, N) = (3, 60)$.

	60													
Jobs	GA_Tabu		Tabu		GA		PACO		MMAX		SA		PSO	
Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)	r_{PACO} (%)	Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)
90	577.5	1092	0.00	179	0.05	2724	0.07	597	0.00	915	1.23	20	1.38	17
80	509.6	975	0.02	159	0.09	2093	0.02	526	0.02	804	1.19	18	1.52	17
70	449.1	866	0.04	140	0.17	1587	0.15	463	0.11	705	1.46	16	1.68	16
60	405.6	774	0.00	127	0.13	1859	0.13	418	0.17	635	1.65	15	1.81	16
50	328.2	645	0.00	101	0.31	1278	0.14	333	0.08	504	1.78	12	2.25	16
40	272.4	522	0.04	84	0.48	1428	0.24	276	0.22	416	2.84	11	2.70	15
30	196.6	384	0.13	59	0.91	986	0.38	197	0.16	295	2.94	9	3.80	15
25	165.1	330	0.18	50	1.91	1366	0.64	166	0.47	249	5.08	8	5.94	15
20	146.8	315	0.63	45	4.20	1511	2.08	148	1.78	222	7.75	7	9.63	14
15	116.9	326	4.68	33	4.00	1081	4.35	107	4.20	161	17.53	6	19.41	14

Table 10. Performance comparisons of algorithms in scenarios with $(M, N) = (3, 80)$.

	80													
Jobs	GA_Tabu		Tabu		GA		PACO		MMAX		SA		PSO	
Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)	r_{PACO} (%)	Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)
90	766.2	2404	0.07	431	0.31	3721	0.21	1429	0.39	2179	1.08	25	1.23	34
80	686.3	2174	0.00	387	0.46	2935	0.15	1278	0.45	1939	1.22	22	1.33	33
70	609.7	1884	0.02	344	0.46	2823	0.17	1133	0.40	1714	1.78	20	1.50	33
60	520.8	1587	0.03	292	0.34	2682	0.17	959	0.70	1449	1.52	17	1.71	32
50	450.7	1377	0.12	252	0.41	3010	0.22	831	0.44	1250	1.65	15	2.01	32
40	354.5	1153	0.05	199	0.76	2283	0.16	654	1.02	983	2.09	12	2.47	31
30	285.8	878	0.08	158	1.30	2269	0.44	521	1.34	779	2.39	10	3.04	31
25	232.6	722	0.04	130	3.16	1692	0.85	427	3.14	639	3.47	9	4.39	30
20	203.8	638	0.16	115	5.36	1485	1.77	376	7.13	562	5.94	8	7.43	30
15	159.3	462	0.07	78	10.29	1609	1.48	253	11.40	378	22.57	6	16.44	30

Table 11. Performance comparisons of algorithms in scenarios with $(M, N) = (3, 100)$.

	100													
Jobs	GA_Tabu		Tabu		GA		PACO		MMAX		SA		PSO	
Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)	r_{PACO} (%)	Yield (%)	C_{GA_Tabu} (min)	t_{GA_Tabu} (s)	r_{Tabu} (%)	t_{Tabu} (s)	r_{GA} (%)	t_{GA} (min)
90	950.5	4379	0.04	839	0.06	5876	0.39	2780	0.28	4236	0.96	30	1.14	59
80	853.9	3970	0.05	757	0.16	4726	0.63	2493	0.34	3788	1.48	27	1.24	58
70	760.4	3484	0.00	674	0.21	4348	0.47	2214	0.23	3349	1.35	24	1.36	57
60	653.2	3375	0.04	580	0.38	4978	0.69	1896	0.57	2865	1.78	21	1.56	57
50	535.1	2450	0.00	475	0.63	3996	0.82	1549	0.67	2336	1.56	17	1.89	56
40	451.1	2046	0.10	397	1.12	4059	1.23	1296	1.21	1953	2.42	14	2.35	56
30	345.0	1540	0.07	300	1.57	2632	1.62	981	1.54	1478	2.11	11	2.93	55
25	306.8	1377	0.33	270	2.32	3063	2.60	882	2.35	1329	4.46	10	3.68	55
20	252.8	1135	0.08	224	5.30	2209	5.61	730	5.54	1097	5.96	9	7.24	54
15	205.7	896	0.16	168	7.81	2238	8.29	533	8.24	803	16.05	7	14.02	54

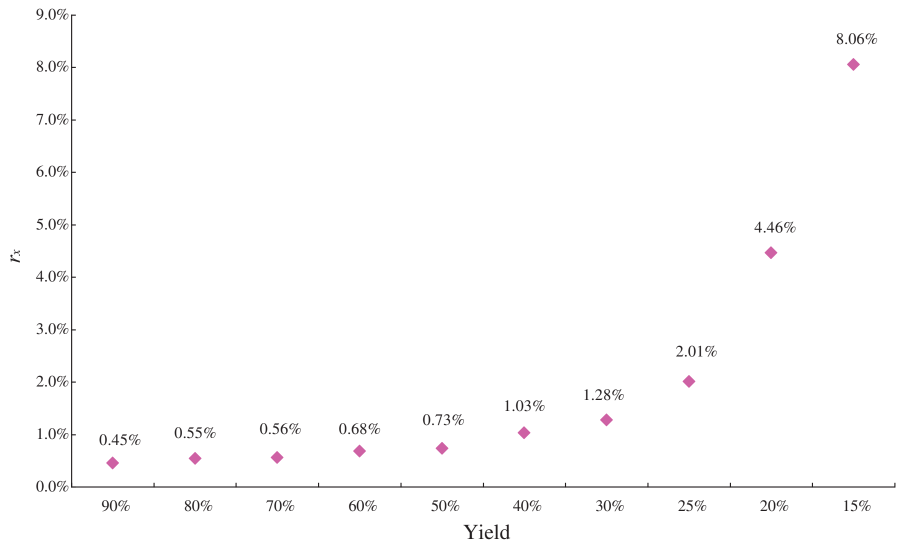

Figure 3. Average of γ_x at various yields.

6.3. *Research uniqueness*

The uniqueness of this research, in comparison to the prior studies on the single-stepper scheduling problem, is explained from the following three perspectives: (1) the scheduling context, (2) the chromosome-decoding and (3) the chromosome solution quality of meta-heuristic algorithms.

The scheduling context of multiple steppers is more complicated than that of single stepper. Namely, the scheduling context of multiple steppers involves two types of decisions: (1) *stepper allocation:* how to allocate jobs to each stepper and (2) *job sequencing:* how to sequence the allocated jobs for each stepper. In contrast, the scheduling context of single stepper involves only one type of decision – how to sequence all the jobs for the stepper.

Such an increased complexity in the scheduling contexts leads to a *chromosome-decoding* issue. As stated, a chromosome originally represents a sequence of all the jobs. Such a chromosome can be directly interpreted as a scheduling solution for the single-stepper context. However, to interpret such a chromosome as a scheduling solution for the multiple-steppers context, we need to develop a decoding scheme – for decoding a chromosome into two types of decisions (*stepper allocation* and *job sequencing*). In summary, for a given chromosome, even though its appearance is exactly the same in the two scheduling

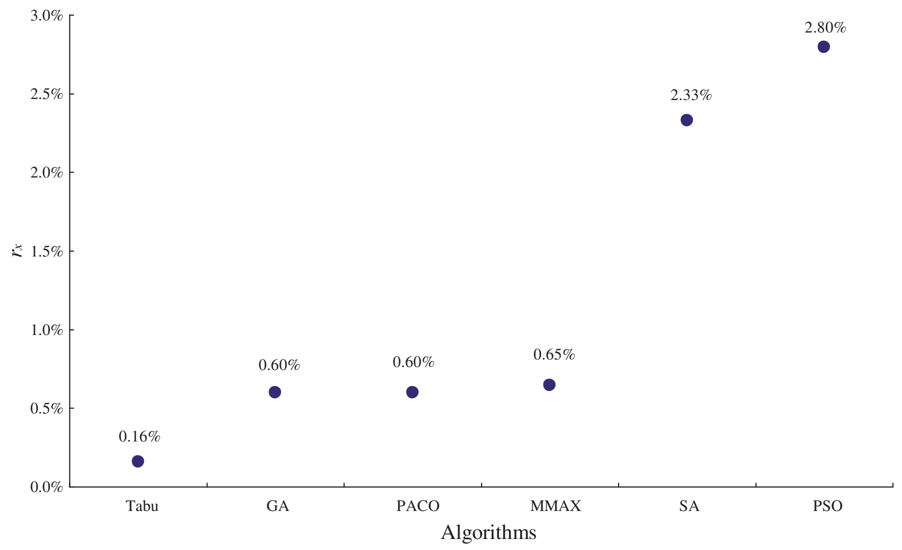

Figure 4. Average of γ_x at various algorithms.

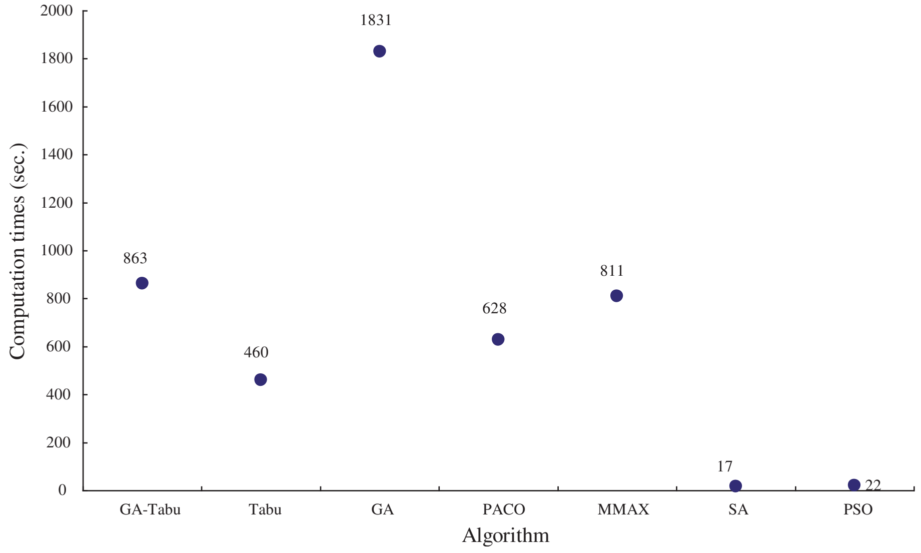

Figure 5. Computation times comparison for the type of algorithms.

contexts, its two implied scheduling decisions are far different.

In this research, several meta-heuristic algorithms are examined in the multiple-steppers context. These algorithms, in terms of algorithmic flow, are essentially the same as those prior ones addressed in the single-stepper context – except the inclusion of the *chromosome-decoding mechanism*. Noticeably, such an inclusion leads to far different semantics in interpreting a chromosome. This lead to that the best-solution chromosome in the single-stepper context is most likely not the best one while it is in the multiple-steppers context. We therefore have to examine the effectiveness of these meta-heuristic algorithms in the multiple-stepper context.

7. Concluding remarks

In-line steppers are the bottleneck of a semiconductor wafer fab. This study examines a problem for the scheduling of N jobs on M in-line steppers in a low-yield scenario. Such a scheduling problem involves two decisions: how to allocate jobs to each stepper, and how to sequence jobs for each stepper. The longest makespan of the M in-line steppers is taken as the performance measure.

A scheduling solution (called a *chromosome*) is represented by a sequence of N jobs. We develop a novel *chromosome-decoding scheme* to interpret a chromosome into its two associated scheduling decisions – job allocation and job sequencing. Such a decoding result can be used to compute the *performance* (also called fitness) of the chromosome.

Based on the chromosome representation and decoding schemes, seven meta-heuristic algorithms adapted from prior studies are developed. These seven algorithms include GA, Tabu, GA–Tabu, SA, M-MMAX, PACO and PSO. Numerical experiments indicate that the GA–Tabu outperforms the other six algorithms in terms of solution quality; and this merit is particularly impressive in low-yield scenarios. In practice, such a scheduling decision is made on every working shift (12 h). The computation time of the GA–Tabu, ranging from a few minutes to less than 1.5 h, is acceptable to practitioners.

One extension of this research is examining the optimal design of port number. The larger the port number, the higher are the tool expenditure and the tool operation costs; yet the capacity loss of steppers is less. Therefore, stepper vendors may need to customise the port design based on the process yields of their customers.

Acknowledgements

This research is financially sponsored by National Science Council, Taiwan, under a research contract NSC 99-2221-E-009-110-MY3.

Notes on contributors

Chie-Wun Chiou received his PhD degree in Industrial Engineering and Management from National Chiao-Tung University, Taiwan. He has been working in the semiconductor industry for over 15 years and possesses several US patents in the semiconductor field. His research interests include capacity planning and production management.

Muh-Cherng Wu received his PhD degree in Industrial Engineering from Purdue University in 1988. He is a Professor in the Department of Industrial Engineering and Management, National Chiao-Tung University, Taiwan. His research interests include supply chain management, production management, in particular in the modelling and analysis of semiconductor manufacturing systems.

References

Bac, F.Q., and Perov, V.L. (1993), 'New Evolutionary Genetic Algorithms for NP-complete Combinatorial Optimization Problems', *Biological Cybernetics*, 69, 229–234.

Chen, T. (2009), 'A Fuzzy-neural Knowledge-based System for Job Completion Time Prediction and Internal Due Date Assignment in a Wafer Fabrication Plant', *International Journal of Systems Science*, 40, 889–902.

Chern, C.C., and Liu, Y.L. (2003), 'Family-based Scheduling Rules of a Sequence-dependent Wafer Fabrication System', *IEEE Transactions on Semiconductor Manufacturing*, 16, 15–25.

Chiou, C.W., and Wu, M.C. (2009), 'GA–Tabu Algorithm for Scheduling In-line Steppers in Low-yield Scenarios', *Expert Systems with Applications*, 36, 11925–11933.

Croce, F.D., Tadei, R., and Volta, G. (1995), 'A Genetic Algorithm for the Job Shop Problem', *Computation Operation Research*, 22, 15–24.

Dabbas, R.M., and Fowler, J.W. (2003), 'A New Scheduling Approach Using Combined Dispatching Criteria in Wafer Fabs', *IEEE Transactions on Semiconductor Manufacturing*, 16, 501–510.

Duwayri, Z., Mollaghasemi, M., Nazzal, D., and Rabadi, G. (2006), 'Scheduling Setup Changes at Bottleneck Workstations in Semiconductor Manufacturing', *Production Planning and Control*, 17, 717–727.

Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Boston, MA: Addison-Wesley.

Holland, J.H. (1975), *Adaptation in Neural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press.

Liao, C.J., Tseng, C.T., and Luarn, P. (2007), 'A Discrete Version of Particle Swarm Optimization for Flowshop Scheduling Problems', *Computers and Operations Research*, 34, 3099–3111.

Michalewicz, Z. (1996), *Genetic Algorithms + Data Structures = Evolution Programs* (3rd ed.), Berlin, Heidelberg, New York: Springer.

Morrison, J.R., and Martin, D.P. (2007), 'Performance Evaluation of Photolithography Cluster Tools', *OR Spectrum*, 33, 375–389.

Nearchou, A.C. (2004), 'The Effect of Various Operators on the Genetic Search for Large Scheduling Problems', *International Journal of Production Economics*, 88, 191–203.

Osman, .H. I, and Potts, C.N. (1989), 'Simulated Annealing for Permutation Flow-shop Scheduling', *OMEGA, The International Journal of Management Science*, 17, 551–557.

Pinedo, M. (2008), *Scheduling: Theory, Algorithms, and Systems* (3rd ed.), New York, NY: Springer.

Quirk, M. (2001), *Semiconductor Manufacturing Technology*, New York, NY: Prentice-Hall.

Rajendran, C., and Ziegler, H. (2004), 'Ant-colony Algorithms for Permutation Flowshop Scheduling to Minimize Makespan/Total Flowtime of Jobs', *European Journal of Operational Research*, 155, 426–438.

Reeves, C.R. (1995), 'A Genetic Algorithm for Flowshop Sequencing', *Computation Operational Research*, 22, 5–13.

Ruiz, R., and Maroto, C. (2006), 'A Genetic Algorithm for Hybrid Flowshops with Sequence Dependent Setup Times and Machine Eligibility', *European Journal of Operational Research*, 169, 781–800.

Sha, D.Y., Hsu, S.Y., Che, Z.H., and Chen, C.H. (2006), 'A Dispatching Rule for Photolithography Scheduling with an On-line Rework Strategy', *Computers and Industrial Engineering*, 50, 233–247.

Wang, L., and Zheng, D.Z. (2003), 'An Effective Hybrid Heuristic for Flowshop Scheduling', *International Journal of Advanced Manufacturing Technology*, 21, 38–44.

Widmer, M., and Hertz, A. (1989), 'A New Heuristic Method for the Flow Shop Sequencing Problem', *European Journal of Operational Research*, 41, 186–193.

Wu, M.C., and Chang, W.J. (2007), 'A Short-term Capacity Trading Method for Semiconductor Fabs with Partnership', *Expert Systems with Applications*, 33, 476–483.

Wu, M.C., and Chang, W.J. (2008), 'A Multiple Criteria Decision for Trading Capacity between Two Semiconductor Fabs', *Expert Systems with Applications*, 35, 938–945.

Wu, M.C., and Chiou, C.W. (2010), 'Scheduling Semiconductor In-line Steppers in New Product/Process Introduction Scenarios', *International Journal of Production Research*, 48, 1835–1852.

Wu, M.C., Chiou, S.J., and Chen, C.F. (2008), 'Dispatching for Make-to-order Wafer Fabs with Machine-dedication and Mask Set-up Characteristics', *International Journal of Production Research*, 46, 3993–4009.

Wu, M.C., Huang, Y.L., Chang, Y.C., and Yang, K.F. (2006), 'Dispatching in Semiconductor Fabs with Machine-dedication Features', *International Journal of Advanced Manufacturing Technology*, 28, 978–984.

Wu, M.C., Jiang, J.H., and Chang, W.J. (2008), 'Scheduling a Hybrid MTO/MTS Semiconductor Fab with Machine-dedication Features', *International Journal Production Economics*, 112, 416–426.

Yang, W.H. (1999), 'Survey of Scheduling Research Involving Setup Times', *International Journal of Systems Science*, 30, 143–155.