A GA-Tabu algorithm for scheduling in-line steppers in low-yield scenarios

Chie-Wun Chiou, Muh-Cherng Wu *

Department of Industrial Engineering and Management, National Chiao Tung University, 1001, Dah-Sheil Road, Hsin-Chu 300, Taiwan, ROC

ARTICLE INFO

Keywords:
Scheduling
Semiconductor
Flow shop
Port capacity constraints
Meta-heuristic algorithm
Genetic algorithm
Tabu search

ABSTRACT

This paper presents a scheduling algorithm for an in-line stepper in low-yield scenarios, which mostly appear in cases when new process/production is introduced. An in-line stepper is a bottleneck machine in a semiconductor fab. Its interior comprises a sequence of chambers, while its exterior is a dock equipped with several ports. The transportation unit for entry of each port is a job (a group of wafers), while that for each chamber is a piece of wafer. This transportation incompatibility may lead to a capacity-loss, in particular in low-yield scenarios. Such a capacity-loss could be alleviated by effective scheduling. The proposed scheduling algorithm, called GA-Tabu, is a combination of a genetic algorithm (GA) and a tabu search technique. Numerical experiments indicate that the GA-Tabu algorithm outperforms seven benchmark ones. In particular, the GA-Tabu algorithm outperforms a prior GA both in solution quality and computation efforts.

1. Introduction

In semiconductor manufacturing, steppers are the most expensive machine and are usually the bottleneck of a wafer fab. Their utilization would significantly affect the throughput of a fab. One way to effectively utilize a stepper is by appropriately scheduling the jobs waiting before it. Scheduling for steppers is thus an important research issue.

Much literature on stepper scheduling has been published. In terms of problem characteristics, they vary in the inclusion of different constraints imposed on steppers—for example, mask setup (Chern & Liu, 2003; Duwayri, Mollaghasemi, Nazzal, & Rabadi, 2006), machine dedication (Wu, Huang, Chang, & Yang, 2006; Wu, Chiou, & Chen, 2007; Wu, Jiang, & Chang, 2008), rework (Sha, Hsu, Che, & Chen, 2006), and cluster tool configuration (Morrison & Martin, 2007). In terms of solution methodology, four types were mostly used: dispatching rules (Dabbas & Fowler, 2003; Ying & Lin, 2009), artificial intelligence (Wu & Chang, 2007, 2008) mathematical programming (Chung & Hsieh, 2008), and meta-heuristic algorithms (Chiang & Fu, 2008).

Significant milestones on stepper scheduling have been established by prior studies. However, most of them implicitly made a high-yield assumption. That is, the production yield is quite high so that each wafer job is a full lot (i.e., carrying 25 wafers). However, in the stage of new product/process introduction, low-yield is quite common. Many small lots (i.e., carrying less than 25 wafers) may appear.

A recent study (Wu & Chiu, 2009) indicated that an in-line stepper, a relatively advanced version of stepper, may loss capacity in a low-yield scenario. Such a capacity-loss would not appear in a high-yield scenario and has been rarely noticed. They proposed a genetic algorithm (GA) for scheduling an in-line stepper. Their experiment results indicated that the GA outperforms four heuristic scheduling rules widely used in practice.

This paper aims to develop a more effective scheduling algorithm for an in-line stepper. The algorithm we proposed is called GA-Tabu, which is a combination of a GA and a tabu search technique. Experiment results indicated that the GA-Tabu outperform seven other meta-heuristic methods, including the GA by Wu and Chiu (2009).

The remainder of this paper is organized as follows. Section 2 introduces the operational mechanism of an in-line stepper. Section 3 describes how to compute the makespan of a job sequence. Section 4 presents the GA-Tabu algorithm. Experiment results are reported in Section 5. Concluding remarks are in the last section.

2. Operational mechanism of an in-line stepper

An in-line stepper is a machine, whose interior comprises a group of chambers, each of which can process one wafer at a time. Its exterior is equipped with a dock for job entry/departure (Fig. 1). Jobs are moved from a neighboring WIP buffer to the dock. The transportation unit between the WIP buffer and the dock is a job (a container carrying 1–25 wafers), but that between the dock and the in-line stepper is a piece of wafer. A transportation incompatibility thus exists between the dock and the in-line stepper.

As shown in Fig. 2, an in-line stepper is composed of two modules—a track and an aligner (Quirk, 2001; Xiao, 2001). Each module
involves various types of chambers. Each type of chamber may be more than one in numbers. In the aligner module, a mask setup time is needed at the exposure chamber, while a wafer from a different job enters the chamber.

The dock of an in-line stepper typically involves four ports. Each port could accommodate only one job container. A job container on the dock must stay at the port until all its wafers have completed processing. This implies that any other jobs cannot access the in-line stepper while all the ports have been occupied. As a result, a capacity-loss may appear in a low-yield scenario, due to the limit of port number.

An example of such a capacity-loss is given below. Consider an in-line stepper that has four ports and 22 chambers. Four jobs (A, B, C, and D) are now on the dock. Job A contains 25 wafers while jobs B, C, and D in total carry only 16 wafers. Suppose the processing sequence is A, B, C, and D. When Job A's last wafer leaves the stepper for the dock, the 16 wafers of jobs B, C, and D would occupy the last 16 chambers of the stepper. This results in a capacity-loss—the first six (22–16) chambers have no new wafers to host. Such a capacity-loss would not occur in a high-yield scenario, in which the total number of wafers in B, C, and D is usually more than 22.

### 3. Makespan evaluation for job sequences

A method, adapted from Ruiz and Maroto (2006), is presented for evaluating the makespan of a job sequence. In the evaluation, we virtually sent each wafer in sequence into the in-line stepper and looked for an available chamber that can finish the wafer at the earliest time. We first give notation and proceed to the makespan evaluation procedure, where a group of functionally identical chambers is called a stage.

#### 3.1. Notation

- **j**: index of job
- **k**: index of wafer
- **i**: index of stage
- **l**: index of chamber
- **a**: index of the exposure chamber
- **p**: total number of ports in the dock
- **n**: total number of jobs to be processed by the in-line stepper
- **M**: total numbers of stages in the in-line stepper
- **m0**: total number of chambers at stage i
- **p0k**: processing time required for chamber l at stage i to process wafer k in job j
- **π**: a job sequence for n jobs, π = [π(1), ..., π(n)]
- **π(j)**: the job in the jth position of a sequence π
- **w(j)**: total number of wafers in job j
- **t_a**: transportation time for uploading a job to the dock
- **t_d**: transportation time for downloading a job from the dock
- **S_{i,l,π(j),k}**: setup time required for chamber l in stage i to process wafer k in job π(j) if j=τ or k=1, then S_{i,l,π(j),k} = 0, otherwise, S_{i,l,π(j),k} = δ_{π(j),π(j-1)}
- **A_{i,l,τ}**: 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,τ} is the last wafer-completion-time before t; while the chamber (i,l) is in operation at t, A_{i,l,τ} is the first wafer-completion-time after t
- **C_{i,π(j),k}**: the completion-time of wafer k in job π(j) at stage i
- **C_{max}(π)**: makespan of job sequence π

#### 3.2. Evaluation procedure

The makespan evaluation procedure is governed by the following four equations:

![Configuration of an in-line stepper](image)
We first introduce the logic flow of the GA-Tabu algorithm, and each stage $i$ gene $C_i$ and the term (sequence) is denoted by $\text{Eq. (1)}$ expresses the completion-time of a particular wafer at each stage $i$. The term $A_{kl} + S_{k(l)j} + C_{l-1}(p,j) + p_{l(1),j,k}$ where $t = C_{l-1}(p,j,k)$ for $1 \leq i \leq M$ (1)

$C_{M-1}(p,j) = C_{M}(p,j) + t_k$ (2)

$C_{M-1}(p,j) + t_k = C_{l-1}(p,j), \text{for} 1 \leq j \leq n - \rho$ (3)

$C_{\text{max}}(p) = C_{M-1}(p,j) + t_k$ (4)

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

Eq. (2) describes the completion-time of job $p(j)$ at stage $M + 1$, where we assume that a waiting-for-process wafer in the port is at stage 0, and a finished wafer in the port is at stage $M + 1$.

Eq. (3) expresses the job arrival/departure relationships for the dock. The equation indicates that job $p(j + \rho)$ in the WIP buffer can be transported to the dock only when job $p(j)$ in the dock has been moved away. Eq. (4) computes the makespan $C_{\text{max}}(p)$.

4. Algorithm

An algorithm called GA-Tabu, a combination of a GA (Holland, 1975) with a tabu search technique, is proposed to solve the in-line stepper scheduling problem. In the algorithm, a chromosome (a job sequence) is denoted by $\pi = [\pi(1), \ldots, \pi(n)]$ where $\pi(j)$, called a gene, represents the job in the $j$th position of sequence $\pi$. The makespan $C_{\text{max}}(p)$ of the chromosome is called its fitness function. We first introduce the logic flow of the GA-Tabu algorithm, and then to describe each major component in the algorithm.

4.1. Logic flow

The logic flow of the proposed algorithm is described by a main procedure named GA-Tabu, in which a sub-procedure named Tabu will be called.

Procedure GA-Tabu

Step 1: Initialization

Set $k = 0$ and $t = 0$. Randomly select $N_p$ chromosomes to form an initial population $P(t)$. Identify the best chromosome $\pi^g$ in $P(t)$. Set $\pi^g = \pi^g_t / \pi^g = \pi^g_t$ is the currently best ever solution; $k$ is the tenure of $\pi^g$ /$

Set tabu_list = $ /tabu_list is a queue list of size $q$ /$

Set $\pi^g = \pi^g_t / \pi^g_t$ is a chromosome called tabu_seed /$

Step 2: Update $P(t+1)$ by GA

Use crossover operators to create a set $N_1(t)$ of $P_{cr} \cdot N_p$ new chromosomes.

Use mutation operators to create a set $N_2(t)$ of $P_m \cdot N_p$ new chromosomes.

Let $S(t) = P(t) \cup N_1(t) \cup N_2(t)$. From $S(t)$, use a selection strategy (the roulette wheel selection method by Michalewicz (1996) to select $N_p$ chromosomes to form $P(t+1)$).

Step 3: Update $\pi^g$ based on $\pi^g_{\text{best}}$, the best chromosome in $P(t+1)$ (1)

From $P(t+1)$, identify the best chromosome $\pi^g_{\text{best}}$.

If $C_{\text{max}}(\pi^g_{\text{best}}) \leq C_{\text{max}}(\pi^g)$ set $k = k + 1$, go to Step 4.

If $C_{\text{max}}(\pi^g_{\text{best}}) > C_{\text{max}}(\pi^g)$, set $k = 0$ /while $\pi^g_{\text{best}}$ is better than $\pi^g$ /$

Set $\pi^g = \pi^g_{\text{best}}$ / update $\pi^g$ by $\pi^g_{\text{best}}$ /$

call Tabu($\pi^g_{\text{best}}, \pi^g$); / further update $\pi^g$ by tabu /$

Step 4: Update $\pi^g$ based on good chromosomes in $P(t+1)$ other than $\pi^g_{\text{best}}$

If $k = 20$, call Tabu($\pi^g_{\text{best}}, \pi^g$), where $\pi^g_{\text{best}}$ is the 2nd best chromosome in $P(t+1)$

If $k = 30$, call Tabu($\pi^g_{\text{best}}, \pi^g$); $\pi^g_{\text{best}}$ is the 3rd chromosome in $P(t+1)$

If $k = 40$, call Tabu($\pi^g_{\text{best}}, \pi^g$); $\pi^g_{\text{best}}$ is the 4th chromosome in $P(t+1)$

Step 5: Update $\pi^g$ and $\pi^g_{\text{best}}$ based on the current tabu seed

If $(k > 40 \&\& \text{mod}(k, 5) = 0)$, call Tabu($\pi^g_{\text{best}}, \pi^g$)

Step 6: Termination Check

If $(k = K$ or $t = T)$, then stop

Otherwise, set $t = t + 1$, go to Step 3.

Procedure Tabu($\pi^g_{\text{best}}, \pi^g$)

Step 1: Create a set $\Omega$ of new chromosomes.

- Based on $\pi^g_{\text{best}}$, apply the steepest descent pairwise interchange (SDPI) (Armour & Buffa, 1963) technique to create a set $\Omega$ of new chromosomes. Each new chromosome $\pi^g$ is created by applying a particular interchange operation, represented by move($\pi$)

Step 2: From $\Omega$, identify the best $q$ chromosome $\pi^{1}_{b}, \ldots, \pi^{q}_{b}$

For $i = 1, q$

If move($\pi^{i}_{b}$) $\notin$ tabu_list, then

Place move($\pi^{i}_{b}$) in the tabu_list;

Set $\pi^{g\text{out}} = \pi^{i}_{b}$;

Replace the worst chromosome in $P(t+1)$ by $\pi^{g\text{out}}$ / update $P(t+1)$ /$

Go to Step 3

Endif

Endfor

Step 3: Update $\pi^g$ and $\pi^g_{\text{best}}$

If $C_{\text{max}}(\pi_{\text{out}}) < C_{\text{max}}(\pi^g)$, set $\pi^g = \pi_{\text{out}}$ and $k = 0$ / update $\pi^g$ /$

If $(\pi_{\text{out}} = \pi^g)$, set $\pi_{\text{out}} = \pi_{\text{out}}$; / update $\pi^g$ /$

Return.

Procedure GA-Tabu is explained below. In Step 1, we create an initial population of chromosomes $P(0)$ and set the initial status of $\pi^g$ and $\pi^g_{\text{best}}$, where $\pi^g$ is the currently best ever chromosome and $\pi^g_{\text{best}}$ is a chromosome called tabu_seed.

In Step 2, we update $P(t)$ by crossover operators, mutation operators and a selection strategy. Crossover and mutation operators are used to create new chromosomes. The existing and new chromosomes are then screened by the selection strategy in order to create $P(t + 1)$ from $P(t)$.

In Step 3, we intend to update $\pi^g$ based on $\pi^g_{\text{best}}$, the best chromosome in $P(t + 1)$. If $\pi^g_{\text{best}}$ is better than $\pi^g$, this implies that $\pi^g_{\text{best}}$ is a good chromosome and its neighborhood could be exploited further. To do so, we call Procedure Tabu to create a new chromosome $\pi_{\text{out}}$ from the neighborhood of $\pi^g_{\text{best}}$. Of the two chromosomes $\pi^g_{\text{best}}$ and $\pi_{\text{out}}$, the better one if eligible is used to update $\pi^g$. If $\pi^g$ is not updated in an iteration $t$, its tenure $k$ is increased by one. The tenure $k$ indicates how many times $\pi^g$ has outperformed $\pi^g_{\text{best}}$ while $P(t)$ is evolving. A large $k$ value implies that update of $\pi^g$ based on current $\pi^g_{\text{best}}$ in $P(t + 1)$ appears to be difficult to a certain extent.

Therefore, in Step 4, some good chromosomes other than $\pi^g_{\text{best}}$ in $P(t + 1)$ are considered to update $\pi^g$. That is, we use the 2nd, 3rd, and 4th best chromosomes in $P(t + 1)$ and call Procedure Tabu to exploit their neighborhood in order to create new chromosomes for possibly replacing $\pi^g$. In essence, both Steps 3 and 4 are intended to update $\pi^g$ through the use of $P(t + 1)$, which evolves based on the GA technique. While the value of $k$ is too large (here we set $k > 40$), we could infer that updating $\pi^g$ through the use of GA is now hard to a certain extent.

To remedy this deficiency, in Step 5, we alternatively use tabu_seed $\pi^g$ and call Procedure Tabu to exploit its neighborhood in order to create new chromosomes for possibly replacing $\pi^g$. The evolution of $\pi^g$ is independent of the GA. Therefore, $\pi^g$ is likely far away from the neighborhood of the chromosomes in $P(t + 1)$. If $\pi^g_{\text{best}}$ is
This characteristic helps the algorithm escape from a local optimal solution obtained by the GA.

Procedure Tabu(π_{in}, π_{out}) is designed to create a new chromosome π_{out}, which is one in the neighborhood of π_{in}. One purpose for obtaining π_{out} is for updating π^t and P(t+1). The input π_{in} has two possible sources: P(t+1) or tabu_seed π^t. The source π^t is designed to evolve through this procedure. Therefore, we additionally use π_{out} to update π^t while π_{in} = π^t. In summary, Procedure Tabu has three functions: updating π^t, π^t and P(t+1).

4.2. Crossover and mutation operators

To create new chromosomes, we use four types of crossover operators and three types of mutation operators. A crossover operator is designed to create a new pair from an existing pair, while a mutation operator is designed 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 (Neareichou, 2004; Wang & Zheng, 2003; Zhang, Wang, & Zheng, 2006).

The four crossover operators are explained below, where parent-1 and parent-2 denote parent chromosomes while child-1 and child-2 denote the created ones.

**Fig. 3. Crossover operators: (a) C1, (b) PMX, (c) LOX, (d) NABEL.**

**Fig. 4. Mutation operators: (a) SWAP, (b) Inverse, (c) Insert.**

**C1 operator:** Referring to Fig. 3a, one randomly selected point is used to divide each parent into two sections (head and tail). To create an offspring (say, child-2), its head is copied from the head of parent-2—a string (3, 5, 6) in this case. Its tail is determined by sequentially referring to the genes of parent-1: only the gene values not in the head of child-2 are eligible to appear in the tail. This results in a string (1, 9, 4, 8, 7, 2) as the tail of child-2.

**LOX operator:** Referring to Fig. 3b, each parent is randomly divided into three sections (head, middle and tail). The middle for parent-1 and parent-2 are (3, 9, 4) and (6, 1, 8) respectively. A
chromosome \( x_{-child-1} \) is created by two steps. First, its \textit{middle} is copied from the \textit{middle} of parent-1. Second, its genes values that appear in the \textit{middle} of parent-2 are replaced by \textit{“H”}. This yields \( x_{-child-1} = (H, H, 3, 9, 4, 5, H, 7, 2) \). We then manipulate \( x_{-child-1} \) by moving \textit{“H”} to the middle and yield a chromosome \( y_{-child-1} = (3, 9, H, H, H, 4, 5, 7, 2) \). Finally, the \textit{middle} of \( y_{-child-1} \) is replaced by that of parent-2; this yield \( child-1 = (3, 9, 6, 1, 8, 2, 4, 7, 2) \).

**PMX operator:** Referring to Fig. 3c, each parent is randomly divided into three sections (\textit{head, middle, and tail}). A new offspring (e.g., \textit{child-1}) is created by the following procedure. The \textit{middle} of \textit{child-1} is created by referring to that of parent-2, which is a string \((1, 8, 2, 7)\). Both the head and tail of \textit{child-1} are created by referring to parent-1. If the gene values in the head/tail sections of parent-1 do not appear in \textit{child-1}, we copy them in the exact positions of \textit{child-1} (e.g., gene values 6 and 3). Finally, for those vacant genes in \textit{child-1}, we place their values by sequentially referring to the unassigned genes in parent-1. This yields \( child-1 = (9, 6, 3, 1, 8, 2, 7, 4, 5) \).

**NABEL operator:** Suppose the two parent chromosomes are \( p_{1} = [p_{2}(1), ... , p_{2}(n)] \) and \( p_{2} = [p_{1}(1), ... , p_{1}(n)] \); and the two children chromosomes to be created are \( c_{1} = [c_{1}(1), ... , c_{1}(n)] \) and \( c_{2} = [c_{2}(1), ... , c_{2}(n)] \). Referring to Fig. 3d, the NABEL operator is designed to set \( c_{1}(i) = p_{2}(p_{1}(i)) \) and \( c_{2}(i) = p_{2}(p_{1}(i)) \). For example, \( p_{2}(2) = 5 \) and \( p_{1}(2) = 4 \), and we can obtain \( c_{1}(2) = 5 \) and \( c_{2}(2) = 4 \).

The three mutation operators are explained below, where \( p_{a} \) denotes the parent chromosome while \( p_{b} \) denotes the child one.

**Swap operator:** Referring to Fig. 4a, we randomly choose two distinct genes in \( p_{a} \) and then swap their gene values to create \( p_{b} \).

**Inverse operator:** Referring to Fig. 4b, we randomly select two cut-off points in \( p_{a} \) to divide it into three sections. Represent \( p_{a} = [p_{a1}, p_{a2}, p_{a3}] \) and \( p_{b} = [p_{a1}, p_{a4}, p_{a5}] \). The inverse operator is designed to create a new chromosome \( p_{b} = [p_{a1}, p_{a2}, p_{a3}] \) where \( p_{a2}(i) = p_{a4}(m + 1 - i) \).

**Insert operator:** Referring to Fig. 4c, we randomly select an insert point and a segment of genes in \( p_{a} \). This would divide \( p_{a} \) into four sections \( p_{a} = [p_{a1}, p_{a2}, p_{a3}, p_{a4}] \) which denote that the insert point is between \( p_{a2} \) and \( p_{a3} \). The selected segment is \( p_{a3} \). To create a new chromosome, the inverse operator moves the selected segment to the insert point position. This would yield a new chromosome \( p_{b} = [p_{a1}, p_{a2}, p_{a3}, p_{a4}] \).

### 4.3. Procedure Tabu

In Step 1 of Procedure Tabu, a set \( \Omega \) of new chromosomes is created by the SDPI technique. Given a chromosome \( p_{in} \), the inverse technique creates a new one by choosing a pair of genes from \( p_{in} \) and interchanges their values. With \( n \) genes in \( p_{in} \), \( \Omega \) would include \( C(n,2) \) new chromosomes. For a new chromosome \( \pi = [\pi(1), ... , \pi(i), ... , \pi(n)] \), where \( \pi(i) \) and \( \pi(j) \) are the two genes that are interchanged. Then, its interchange operation \( move(\pi) \) is represented \( \{\pi(i), \pi(j)\} \).

In Step 2 of the procedure, out of \( \Omega \), only the best possible chromosome created by a \textit{“new”} move is eligible for updating \( \pi^{*} \), \( \pi^{*} \), and \( P(t + 1) \). Herein, a \textit{“new”} move is one, currently not in the tabu_list. This implies that the tabu_list is designed to record “good” and “new” moves. Such a design is to avoid a cyclic creation of the same chromosome. This would keep \( \pi^{*} \) and \( \pi^{*} \) away from being trapped into a local optima.

### 5. Numerical experiments

By numerical experiments, we justify the effectiveness of the proposed GA-Tabu algorithm. The in-line stepper is assumed to have four ports, 14 stages and 21 chambers. The operation time at each chamber \( i \) follows a uniform distribution \([a_{i}, b_{i}]\) (Table 1).

A mask setup is always required for the exposure chamber while it turns to process a new job’s wafer, and the mask change time is a constant (1.0 min). Parameters in the GA-Tabu are set as follows: \( P = 100 \), \( P_{cr} = 0.8 \), \( P_{mu} = 0.2 \), \( q = 7 \), \( K = 3000 \), \( T = 100,000 \).

#### 5.1. Test cases

To model the process yield in experiments, we use a truncated binomial distribution (TBD). The TBD implies that the job size is governed by a binomial distribution; however, the jobs carrying no wafer are moved away from the fab.

We use \((N, Y)\) to represent a test case, where \( N \) represents the number of jobs and \( Y \) represents the average yield. In our experiments, \( N \) involves five options ranging from 20 to 100 jobs while \( Y \) involves 10 options ranging from 15% to 90% (Table 2–6). That is, each algorithm has 50 test cases, and each test case is justified by 15 replicates.

#### 5.2. Benchmark algorithms

For each test case, we compare the GA-Tabu with seven other algorithms: optimum heuristic rule (OHR), simulated annealing (SA) by Osman and Potts (1989), GA by Wu and Chiou (in press), tabu search (TS) by Widmer and Hertz (1989), two ant colony algorithms (ACO) by Rajendran and Ziegler (2004), and particle swarm optimization (PSO) by Liao, Tseng, and Luarn (2007). The OHR algorithm denotes taking the best result out of three heuristic scheduling rules: SPT (shortest job processing time), LPT (longest job processing time), and NEH (Nawaz, Enscorre, & Ham, 1983). The two ACOs are respectively called MMAX and PACO.

For each algorithm in each test case, the average makespan of the 15 replicates is taken as the performance. Define the average makespan of each algorithm as follows: \( CGA_{-tabu} \) for the GA-Tabu, \( C_{OHR} \) for the OHR, \( C_{SA} \) for the SA, \( C_{GA} \) for the GA, \( C_{TS} \) for the TS, \( C_{MMAX} \) for the MMAX, \( C_{PACO} \) for the PACO, and \( C_{PSO} \) for the PSO. The computation time for each algorithm is defined accordingly; for example, that for the GA-Tabu is defined as \( T_{GA_{-tabu}} \).
## Table 3
Makespan comparison at different binomial yield scenarios for job 40.

<table>
<thead>
<tr>
<th>Jobs</th>
<th>GA_Tabu</th>
<th>OHR</th>
<th>Random</th>
<th>SPT</th>
<th>LPT</th>
<th>NEH</th>
<th>C_A</th>
<th>C_max</th>
<th>GA</th>
<th>SA</th>
<th>Tabu</th>
<th>MMAX</th>
<th>PACO</th>
<th>PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yield (%)</td>
<td>C_GA-tabu (min)</td>
<td>t_GA-tabu (s)</td>
<td>C_GA (min)</td>
<td>t_GA (s)</td>
<td>C_SA (min)</td>
<td>t_SA (s)</td>
<td>C_max (min)</td>
<td>t_max (s)</td>
<td>C_PACO (min)</td>
<td>t_PACO (s)</td>
<td>C_PSO (min)</td>
<td>t_PSO (s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>1155.6 381</td>
<td>1170.0 381</td>
<td>1156.7 381</td>
<td>1158.7 381</td>
<td>1155.6 381</td>
<td>1158.0 381</td>
<td>1155.6 381</td>
<td>1155.9 381</td>
<td>1158.0 381</td>
<td>1155.9 381</td>
<td>1158.0 381</td>
<td>1155.9 381</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>1001.0 333</td>
<td>1015.7 333</td>
<td>1011.9 333</td>
<td>1004.4 333</td>
<td>1008.0 333</td>
<td>1004.4 333</td>
<td>1001.0 333</td>
<td>1001.0 333</td>
<td>1001.0 333</td>
<td>1004.4 333</td>
<td>1008.0 333</td>
<td>1001.0 333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>901.3 294</td>
<td>914.7 294</td>
<td>910.7 294</td>
<td>905.4 294</td>
<td>906.8 294</td>
<td>905.4 294</td>
<td>901.3 294</td>
<td>901.3 294</td>
<td>901.3 294</td>
<td>901.3 294</td>
<td>901.3 294</td>
<td>901.3 294</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>773.7 276</td>
<td>787.9 276</td>
<td>781.5 276</td>
<td>783.1 276</td>
<td>783.1 276</td>
<td>773.7 276</td>
<td>787.9 276</td>
<td>781.5 276</td>
<td>783.1 276</td>
<td>787.9 276</td>
<td>781.5 276</td>
<td>783.1 276</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>677.8 261</td>
<td>691.0 261</td>
<td>685.7 261</td>
<td>681.8 261</td>
<td>681.8 261</td>
<td>677.8 261</td>
<td>691.0 261</td>
<td>685.7 261</td>
<td>681.8 261</td>
<td>691.0 261</td>
<td>685.7 261</td>
<td>681.8 261</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>519.2 182</td>
<td>531.9 182</td>
<td>527.2 182</td>
<td>523.8 182</td>
<td>523.8 182</td>
<td>519.2 182</td>
<td>531.9 182</td>
<td>527.2 182</td>
<td>523.8 182</td>
<td>531.9 182</td>
<td>527.2 182</td>
<td>523.8 182</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>405.9 150</td>
<td>419.0 150</td>
<td>413.8 150</td>
<td>411.3 150</td>
<td>411.3 150</td>
<td>405.9 150</td>
<td>419.0 150</td>
<td>413.8 150</td>
<td>411.3 150</td>
<td>419.0 150</td>
<td>413.8 150</td>
<td>411.3 150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>346.8 150</td>
<td>367.3 150</td>
<td>358.8 150</td>
<td>356.1 150</td>
<td>356.1 150</td>
<td>346.8 150</td>
<td>367.3 150</td>
<td>358.8 150</td>
<td>356.1 150</td>
<td>367.3 150</td>
<td>358.8 150</td>
<td>356.1 150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>247.3 104</td>
<td>290.7 104</td>
<td>268.2 104</td>
<td>267.2 104</td>
<td>267.2 104</td>
<td>247.3 104</td>
<td>290.7 104</td>
<td>268.2 104</td>
<td>267.2 104</td>
<td>290.7 104</td>
<td>268.2 104</td>
<td>267.2 104</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 4
Makespan comparison at different binomial yield scenarios for job 60.

| Jobs | GA_Tabu | OHR | Random | SPT | LPT | NEH | CG | C_{GA} (min) | \gamma_{GA} (%) | t_{GA} (s) | C_{SA} (min) | \gamma_{SA} (%) | t_{SA} (s) | C_{Tabu} (min) | \gamma_{Tabu} (%) | t_{tabu} (s) | C_{MMAX} (min) | \gamma_{MMAX} (%) | t_{max} (s) | C_{PACO} (min) | \gamma_{PACO} (%) | t_{paco} (s) | C_{PSO} (min) | \gamma_{PSO} (%) | t_{PSO} (s) |
|------|---------|-----|--------|-----|-----|-----|----|-------------|----------------|---------|-------------|----------------|---------|----------------|----------------|------------|----------------|----------------|---------|----------------|----------------|---------|----------------|----------------|---------|----------------|----------------|---------|
| 60   | 1707.1  | 961 | 1726.0 | 1746.0 | 1714.3 | 1710.2 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 |
| 90   | 1707.1  | 961 | 1726.0 | 1746.0 | 1714.3 | 1710.2 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 |
| 50   | 1707.1  | 961 | 1726.0 | 1746.0 | 1714.3 | 1710.2 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 |
| 30   | 1707.1  | 961 | 1726.0 | 1746.0 | 1714.3 | 1710.2 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 |
| 15   | 1707.1  | 961 | 1726.0 | 1746.0 | 1714.3 | 1710.2 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 | 1707.1 | 0.00 | 1670 |

### Table 5
Makespan comparison at different binomial yield scenarios for job 80.

<p>| Jobs | GA_Tabu | OHR | Random | SPT | LPT | NEH | CG | C_{GA} (min) | \gamma_{GA} (%) | t_{GA} (s) | C_{SA} (min) | \gamma_{SA} (%) | t_{SA} (s) | C_{Tabu} (min) | \gamma_{Tabu} (%) | t_{tabu} (s) | C_{MMAX} (min) | \gamma_{MMAX} (%) | t_{max} (s) | C_{PACO} (min) | \gamma_{PACO} (%) | t_{paco} (s) | C_{PSO} (min) | \gamma_{PSO} (%) | t_{PSO} (s) |
|------|---------|-----|--------|-----|-----|-----|----|-------------|----------------|---------|-------------|----------------|---------|----------------|----------------|------------|----------------|----------------|---------|----------------|----------------|---------|----------------|----------------|---------|----------------|----------------|---------|----------------|----------------|---------|
| 80   | 2274.0  | 1851 | 2297.0 | 2305.3 | 2277.7 | 2282.8 | 2277.7 | 0.00 | 2502 | 2288.4 | 0.63 | 49 | 2274.0 | 0.00 | 1210 | 2274.4 | 0.02 | 2084 | 2274.2 | 0.01 | 1474 | 2289.0 | 0.66 | 34 |
| 90   | 2274.0  | 1851 | 2297.0 | 2305.3 | 2277.7 | 2282.8 | 2277.7 | 0.00 | 2502 | 2288.4 | 0.63 | 49 | 2274.0 | 0.00 | 1210 | 2274.4 | 0.02 | 2084 | 2274.2 | 0.01 | 1474 | 2289.0 | 0.66 | 34 |
| 50   | 2274.0  | 1851 | 2297.0 | 2305.3 | 2277.7 | 2282.8 | 2277.7 | 0.00 | 2502 | 2288.4 | 0.63 | 49 | 2274.0 | 0.00 | 1210 | 2274.4 | 0.02 | 2084 | 2274.2 | 0.01 | 1474 | 2289.0 | 0.66 | 34 |
| 30   | 2274.0  | 1851 | 2297.0 | 2305.3 | 2277.7 | 2282.8 | 2277.7 | 0.00 | 2502 | 2288.4 | 0.63 | 49 | 2274.0 | 0.00 | 1210 | 2274.4 | 0.02 | 2084 | 2274.2 | 0.01 | 1474 | 2289.0 | 0.66 | 34 |
| 15   | 2274.0  | 1851 | 2297.0 | 2305.3 | 2277.7 | 2282.8 | 2277.7 | 0.00 | 2502 | 2288.4 | 0.63 | 49 | 2274.0 | 0.00 | 1210 | 2274.4 | 0.02 | 2084 | 2274.2 | 0.01 | 1474 | 2289.0 | 0.66 | 34 |</p>
<table>
<thead>
<tr>
<th>Yield (%)</th>
<th>90</th>
<th>70</th>
<th>50</th>
<th>30</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makespan (s)</td>
<td>2826.8</td>
<td>2257.0</td>
<td>736.0</td>
<td>610.6</td>
<td>497.0</td>
</tr>
<tr>
<td>GA-Tabu</td>
<td>2826.8</td>
<td>2257.0</td>
<td>736.0</td>
<td>610.6</td>
<td>497.0</td>
</tr>
<tr>
<td>GA</td>
<td>2826.8</td>
<td>2257.0</td>
<td>736.0</td>
<td>610.6</td>
<td>497.0</td>
</tr>
<tr>
<td>SA</td>
<td>2826.8</td>
<td>2257.0</td>
<td>736.0</td>
<td>610.6</td>
<td>497.0</td>
</tr>
<tr>
<td>MMAS</td>
<td>2826.8</td>
<td>2257.0</td>
<td>736.0</td>
<td>610.6</td>
<td>497.0</td>
</tr>
<tr>
<td>SPT</td>
<td>2826.8</td>
<td>2257.0</td>
<td>736.0</td>
<td>610.6</td>
<td>497.0</td>
</tr>
<tr>
<td>Random</td>
<td>2826.8</td>
<td>2257.0</td>
<td>736.0</td>
<td>610.6</td>
<td>497.0</td>
</tr>
</tbody>
</table>

From Tables 2–6, we could see that all γ ranges from 0% to 10%. This indicates that the GA-Tabu outperforms all the benchmark algorithms, in terms of solution quality. Notice that this merit appears more impressive in low-yield scenarios than in high-yield scenarios. The GA-Tabu is relatively computationally extensive. However, compared to the TS and the GA (the 2nd and 3rd best ones in terms of solution quality), the GA-Tabu is faster computationally.

The experiment results indicate that the proposed GA-Tabu has its merit—in particular in a low-yield scenario. With in-line steppers as the bottleneck of a fab, even a 1% increase in the in-line stepper throughput would have a substantial positive impact on gross margins.

6. Concluding remarks

This study examines a scheduling problem for a semiconductor in-line stepper, with makespan as the performance criterion. We propose a meta-heuristic algorithm, called GA-Tabu, to solve the problem. Seven other scheduling algorithms are compared with the GA-Tabu by numerical experiments. Experiment results indicated that the GA-Tabu outperforms all the benchmarks in terms of solution quality. This merit is in particular more impressive in low-yield scenarios.

One extension to this research is the scheduling of two or more in-line steppers. Such an extension would involve one more decision-making—how to allocate jobs to each in-line stepper. Another extension is the configuration design for an in-line stepper, for example, determining the optimum number of ports.

References


