Scheduling Unrelated Parallel Machines in Semiconductor Manufacturing by Problem Reduction and Local Search Heuristics

Chih-Wei Chen

Abstract—**We investigate a difficult scheduling problem in a semiconductor manufacturing process that seeks to minimize the number of tardy jobs and makespan with sequence-dependent setup time, release time, due dates and tool constraints. We propose a mixed integer programming (MIP) formulation which treats tardy jobs as soft constraints so that our objective seeks the minimum weighted sum of makespan and heavily penalized tardy jobs. Although our polynomial-sized MIP formulation can correctly model this scheduling problem, it is so difficult that even a feasible solution can not be solved efficiently for small-scale problems. We then propose a technique to estimate the upper bound for the number of jobs processed by a machine, and use it to largely reduce the size of the MIP formulation. In order to effectively handle real-world large-scale scheduling problems, we propose an efficient dispatching rule that assigns a job of the earliest due date to a machine with least recipe changeover (EDDLC) and try to reoptimize the solution by some local search heuristics which involves interchange, translocation and transposition between assigned jobs. Our computational experiments indicate that EDDLC and our proposed reoptimization techniques are very efficient and effective. In particular, our method usually give solutions very close to the exact optimum for smaller scheduling problems, and can also produce good solutions for scheduling up to 200 jobs on 40 machines within 10 minutes.**

Index Terms—Scheduling, mixed integer programming, dispatching rule, local search.

I. Introduction

IN the semiconductor manufacturing, wafers must undergo hundreds of processing steps before becoming final products. In a clean wafer fab, expensive environment control equipments keep the relative humidity at a fixed level and wipe out particles in the air, to make sure products of high quality are produced via reliable manufacturing processes. The manufacturing machines are also very expensive which can cost millions of dollars. For these reasons, the performance of the manufacturing process is very important. In particular, better manufacturing plans that can produce more products in shorter time are preferred.

To process large amount of wafers, a wafer fab often has several *tool groups*. The machines in a tool group have similar properties and can process identical recipes. A *recipe* is a formula for mixing or preparing some materials for processing. A recipe change takes a *setup* time. In practice, even if a scheduling plan of jobs is planned in advance, it often has to be altered in order to deal with some urgent assignments (usually called as "*super hot lots*"). As a result, a scheduling plan especially requires to be updated in short time to guarantee a smooth production plan. In general, a good manufacturing plan should schedule necessary recipe changes in a good order, and be able to quickly updated in response to urgent jobs, so that more products can be produced in shorter time.

This research focuses on a scheduling problem over a one-stage production system. A job is processed after it is released from its preceding work station. We assume the following information are given: (1) the release time, processing time, due date, and required recipe for each job; (2) the available time, setup time and recipes that can be processed. The machines are assumed to be in a perfect condition. Transportation time is neglected, thus all jobs are assumed to arrive at their next work stations immediately after they finish their current processes. We schedule to minimize the number of tardy jobs with minimal makespan. The problem investigated in this paper is a very complex scheduling problem. To the best of our knowledge, there exists no method that can give optimal solution to this problem within polynomial time.

The structure of this paper is as follows: Section 2 reviews and summarizes related literature in scheduling parallel machines; Section 3 gives mathematical formulations for our problem and proposes techniques to reduce the size of original formulations; Section 4 introduces our proposed heuristics including dispatching rules and reoptimization mechanisms; computational experiments are conducted with analysis in Section 5; and Section 6 concludes this paper.

II. LITERATURE REVIEW

A. *Introduction for Scheduling Problem*

Scheduling problems in manufacturing processes assigns jobs to machines for some specific objective. Since different job-machine assignments give different schedules and there are too many feasible arrangements, such problems are usually very difficult to solve, both theoretically and empirically.

Scheduling problems can be represented in the form $\alpha|\beta|\gamma$, where α describes the machine environment, such as single machine, parallel machine, job shop and flow shop; β describes the process constraints, such as release time, due date, batch processing and sequence-dependent setup time.; and γ contains the information about performance measure to be considered, such as makespan, weighted total tardiness and regular cost function.

Zhu and Wilhelm [1] classified scheduling problems by some characteristics such as machine characteristic, release time, due date, one-stage or multi-stages, batching processing and setup time as summarized in Tables I, II and III.

To deal with scheduling problem, a number of solution methods have been proposed, including optimizing methods (Branch and Bound (B&B), branch and cut (B&C), Dynamic Programming (DP), and Mixed Integer Programming (MIP) solvers), hybrids (methods that combine B&B, DP, or MIP solvers with a heuristic), and heuristics (metaheuristics such as genetic algorithms (GA), simulated annealing (SA), tabu search (TS), decomposition, dispatching rules, simulation and list scheduling).

TABLE I. CLASSIFICATION OF MACHINE CONFIGURATION

1	Single machine.
Fm	Flow shop.
FFc	Flexible flow shop with *c* stages in series.
FJc	Flexible job shop with *c* work centers.
HFc	Hybrid flow shop with *c* stages in series, each with a set of unrelated machines in parallel.
Jm	Job shop in which each job has its own predetermined routing.
Pm	Identical machines in parallel.
Qm	Uniform machines in parallel, each operating at a different speed.
Rm	Unrelated machines in parallel, each with a unique processing time for a job.

TABLE II. CLASSIFICATION OF PROCESSING RESTRICTIONS

block	Blocking can occur in a flow shop because buffers have limited capacities.
brbdwn	Breakdown or shutdown of machines.
bsij(k)	Sequence-dependent setup time for batch *j* immediately after batch *i* (on machine *k*).
dj	Jobs have due dates.
d	All jobs have a common due date.
Mj	Not all machines in parallel are capable of processing job *j*.
nwt	Jobs cannot wait between operations in a flow shop.
prmp	Jobs can be preempted.
prec	Precedence constraints relate job.
prmu	A permutation sequence is used in a flow shop.
rj	Jobs have known release times.
recrc	Jobs may recirculate to be processed on the same machine several times.
sij(k)	Sequence-dependent setup time for job *j* immediately after job *i* (on machine *k*).

TABLE III. CLASSIFICATION OF OBJECTIVE

$C_{\max}$	Makespan.
$L_{\max}$	Maximum lateness.
$\sum(w_j)C_j$	Total (weighted) completion time.
$\sum(w_j)T_j$	Total (weighted) tardiness.

$\sum (w_j) U_j$	(Weighted) number of tardy jobs.
$\sum E_j + \sum T_j$	Total earliness/tardiness.
$\sum w' E_j + \sum w'' T_j$	Total weighted earliness/tardiness with the same earliness penalty for all jobs and the same tardiness penalty for all jobs.
$\sum w'_j E_j + \sum w''_j T_j$	Total weighted earliness/tardiness with arbitrary earliness and tardiness penalties for all jobs.
$\sum s_{ij}$, $\sum s_{ijk}$, $\sum bs_{ij}$ or $\sum bs_{ijk}$	Total setup time with respect to s_{ij}, s_{ijk}, b_{sij} or b_{sijk}.
Π	Minimize cost.

B. Complexity of scheduling parallel machines with sequence dependent setup time

Pinedo [2] showed that a single machine scheduling problem that considers sequence dependent setup time to minimize the makespan is equivalent to the Traveling Salesman Problem (TSP), which is NP-hard. More specifically, here in this paper, we investigate a problem which considers parallel machines and constraints of sequence dependent setup time, release time and due date. Therefore, our problem is also NP-hard.

C. Dispatching rules

Dispatching rules are set of simple and intuitive decision rules based on greedy ideas. They are usually used to select the next job to be processed from a set of jobs to suitable machines, and are widely used for solving real-world scheduling problems. Most scheduling systems make scheduling decisions based on dispatching rules which integrate several real time decision rules. Since the scheduling problems are so complex, different dispatching rules can be proposed based on different aspects of the complex scheduling problems, and there usually exist no dominating dispatching rules that can always give better schedules than others in all situations. Nevertheless, there do exist some popular dispatching rules that usually produce good schedules.

Most of the dispatching rules are based on greedy concepts. In general, dispatching rules suggest the priority of the jobs to be processed for each machine. The job with the highest priority is selected to process first. If there exist several jobs of the same priority, the jobs will be selected by random or some auxiliary rules. Blackstone *et al.* [3] categorized the dispatching rules into four classes by the criteria of job or machine selections: (1) rules based on processing time: LPT [4], SPT [4]; (2) rules based on due dates: EDD (earliest due date) [3]; (3) rules other than processing time and due dates:

FIFO[3], FOL [5] , LFJ [2], LFM [2]; (4) hybrid rules involving two or more rules of previous classes: ATCS [6], CR [4], SL [4], WIPQ [7]. Note that no dispatching rules can guarantee to give a schedule of no tardy jobs.

D. Parallel machine scheduling

To the best of our knowledge, very few research up to the present provide mathematical formulation to parallel machine scheduling problem with release time, due date and sequence dependent setup time. Most research on scheduling jobs over parallel machines focus on heuristics.

Ovacik and Uzsoy [8] solved $Pm|r_j,s_{ij}|L_{\max}$ by rolling horizon heuristic (RHP). Schutten [9] presented a list scheduling algorithm for $Pm|r_j,s_{ij}|L_{\max}$. Kurz and Askin [10] formulated $Pm|r_j,s_{ij}|C_{\max}$ as an MIP. Kim [11] solved $Pm|s_{ij}|\sum w_jT_j$ by heuristic algorithm combining dispatching rule (EDD) and tabu search. Balakrishnan [12] provided an MIP for $Qm|r_j,s_{ij}|\sum w_j'E_j+\sum w_j''T_j$. Sivrikaya-Serifoglu and Ulusoy [13] used genetic algorithms to solve $Qm|r_j,s_{ij}|\sum w_j'E_j+\sum w_j''T_j$. Bilge *et al.* [14] applied tabu search for $Qm|r_j,s_{ijk}|\sum T_j$. Arzi and Raviv [15] suggested several dispatching rules for $Rm|r_j,s_{ijk}|$throughput, $\sum s_{ijk}$ and work in process (WIP). Bank and Werner [16] suggested constructive and iterative algorithms to solve $Rm|r_j,d|\sum w_j'E_j+\sum w_j''T_j$. Kim and Shin [17] used tabu search to solve $Rm|r_j,s_{ij}|L_{\max}$. Logendran *et al.* [18] presented several tabu search algorithms with different initial solution finding mechanisms and search mechanisms to investigate $Rm|r_j,s_{ij}|\sum w_jT_j$. They observed that the solution quality and efficiency may be affected by different search mechanisms but not by different initial solution finding mechanisms.

In this paper, we focus on the scheduling problem of type $Rm|r_j,d_j,s_{ijk},M_j|\sum wU_j+w'C_{\max}$. According to the literature, this problem is NP-hard. Most previous researches in this topic suggest the use of heuristics such as tabu search, or dispatching rules to deal with parallel machine scheduling problems with release time and due date. We will first propose an MIP formulation for this problem and then reduce its size by some heuristics in Section 3. Then, we will also propose several efficient and effective heuristics for this problem in Section 4.

III. PROBLEM DEFINITION AND MIP FORMULATION

A. Problem description

Suppose the jobs and tool groups are given. A tool group means a set of machines that can process the same recipes but may spend different processing times, and the machines in different tool groups may be able to process some identical recipes. Each job has a recipe to be processed. There is a setup time on a machine to change recipes for consecutively processing jobs of different recipes. A job can only be processed after its release time, and has to be completed before its due date. The objective is to minimize number of tardy jobs with minimal makespan.

B. Assumptions

Detailed assumptions are as follows: (1) each job has a recipe to be processed, (2) job preemption or cancellation is not allowed, (3) each recipe can be performed by one or more different tool groups, (4) setup times only depend on recipes and machines, (5) each recipe has its own definite processes and operational time, and the time only depends on the machine, and (6) one operation can be processed on one machine at the same time.==>

C. Mixed Integer Programming Model Formulation

The notations of our model are listed as follows:

j	the index of job;
m	the index of machine;
p	the index of position;
r	the index of recipe;
J_r	the set of jobs have recipe r;
J_m'	the set of jobs which can be processed on machine m;
M_j	the set of machines which can process job j;
$p_{j,m}$	processing time of job j on machine m;
$t_{m,r,r'}$	setup time from recipe r to recipe r' on machine m;
r_j	release time of job j;
d_j	due date of job j;
MT_m	available time of machine m;
$Y_{m,0,r}$	=1 if the initial recipe of machine m is recipe r, 0 otherwise;
Np	number of jobs which are allowed to be processed by one machine;
H_1	a very large positive number;
H_2	a very large positive number, but $H_1 >> H_2$;
$X_{j,m,p}$	=1 if job j is assigned to position p on machine m, 0 otherwise;
$Y_{m,p,r}$	=1 if position p on machine m is assigned to process recipe r, 0 otherwise;
Zm,p,r,r'	=1 if position p on machine m is assigned to process recipe r and position p+1 on machine m is assigned to process recipe r', 0 otherwise;
U_j	=1 if job j is not completed before its due date, 0 otherwise;
$S_{m,p}$	starting time of position p on machine m;
$C_{m,p}$	finishing time of position p on machine m;
C_{max}	Makespan.

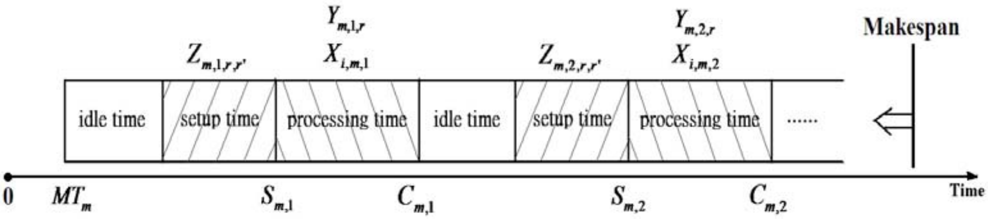

Fig. 1. A possible schedule and its corresponding variables

We give our first mixed integer programming model as follows:

Objective:

$$Min \quad Z = \sum_j H_2 U_j + C_{\max} \tag{1}$$

Constraints:

$$S_{m,p} + \sum_{j \in J'_m} p_{j,m} X_{j,m,p} = C_{m,p} \quad \forall m, p \tag{2}$$

$$Y_{m,p,r} \geq \sum_{j \in J_r} X_{j,m,p} \quad \forall m, p, r \tag{3}$$

$$Z_{m,p,r,r'} \geq Y_{m,p-1,r} + Y_{m,p,r'} - 1 \quad \forall m, p, r, r' \tag{4}$$

$$C_{m,p} + \sum_r \sum_{r'} t_{m,r,r'} Z_{m,p+1,r,r'} \leq S_{m,p+1} \quad \forall m, p \tag{5}$$

$$MT_m + \sum_r \sum_{r'} t_{m,r,r'} Z_{m,1,r,r'} \leq S_{m,1} \quad \forall m \tag{6}$$

$$\sum_{j \in J'_m} X_{j,m,p} \geq \sum_{j \in J'_m} X_{j,m,p+1} \quad \forall m, p \tag{7}$$

$$\left(H_1 + r_j\right) X_{j,m,p} - S_{m,p} \leq H_1 \quad \forall j, m, p \tag{8}$$

$$C_{m,p} + \left(H_2 - d_j\right) X_{j,m,p} - H_2 U_j \leq H_2 \quad \forall j, m, p \tag{9}$$

$$C_{\max} \geq C_{m,Np} \quad \forall m \tag{10}$$

$$\sum_{j \in J'_m} X_{j,m,p} \leq 1 \quad \forall m, p \tag{11}$$

$$\sum_{m \in M_j} \sum_p X_{j,m,p} = 1 \quad \forall j \tag{12}$$

Fig. 1 illustrates a possible schedule on a specific machine. Equation (1) first minimizes the number of jobs exceeding due date, and then minimizes the makespan. For a job in position *p* on machine *m*, equation (2) defines how to calculate its finishing time by its starting time and processing time. Equation (3) defines whether job *j* belongs to recipe *r*, and is assigned to position *p* on machine *m*. Equation (4) defines whether there is a recipe changeover time on machine *m* between positions *p*-1 and p. Equation (5) forces the job on position p+1 to start after the job on position p finishes, for any machine m. Equation (6) is the application of (5) for position 1. Equation (7) is a consecutive job assignment constraint, which forces jobs to be assigned consecutively, starting from position 1. In other words, for any machine, if its position p+1 has been assigned a job, then all previous positions 1,…,p has to be nonempty. Equation (8) is active when $X_{j,m,p}$ =1, which reflects the relationship between release time of job j and the operation assigned to position p on machine m. Equation (9) is active when $X_{j,m,p}$ =1 and U_j =0, which reflects the relationship between due date of job j and the operation assigned to position p on machine m. Equation (10) imposes that makespan should be greater or equal to the finishing time of all jobs. Equation (11) represents the assumption that no more than one operation can be assigned to a machine simultaneously. Equation (12) imposes that a job can only be assigned to one position of one machine. For a scheduling problem involves N jobs, M machines, R recipes, and P positions, there will be $MP(N+R+R^2)+N$ binary variables, $2MP+1$ real-valued variables and $M(2+4P+2NP+PR+PR^2)+N$ constraints in the MIP formulation.

D. A Heuristic to Reduce the Size of MIP

For the complete MIP formulation introduced in previous section, the number of positions is set to be the number of jobs, which would be the worst case and only happens in very rare situations. Such an upper bound setting results in much waste in both storage and computational time, because no single machine will process all jobs while all other machines are idle in practice. In other words, the complete formulation usually leads to a huge MIP, usually unsolvable in short time, even when the number of jobs, machines, and recipes are not too large.

On the other hand, if the manufacturing process is very stable and steady, then the number of jobs to be processed on each machine may be about the same to each other and close to the number of jobs divided by the number of machines. In general, we should try to estimate a good and smaller upper bound on the number of positions that a machine may be capable of processing, which would in turn help us derive a smaller MIP formulation.

One may observe that a good schedule would balance the loads for each machine with few recipe changeovers. In order to reduce the number of recipe changeovers, we should try to process the jobs of the same recipe as consecutively as possible on the same machine. Based on this observation, we suggest to use equation (13) for estimating Np, the upper bound on the number of job positions for each machine, where n_i is the number of processible jobs by tool group i, m_i is the number of machines belonging to tool group i, $p_{i\min}$ and $p_{i\max}$ represent the shortest and longest processing times of all operations that can be performed by tool group i, respectively.

$$Np = \max_i \left\lceil \frac{n_i}{1+(m_i-1)\, p_{i\min} / p_{i\max}} \right\rceil \tag{13}$$

Theoretically, an extreme case happens when we put all the jobs of the shortest processing time into a single machine, and use the others (i.e. m_i -1 many) to process all the other jobs of the longest processing time. With sufficient job sources, when the machine finishes processing one job of the shortest processing time, the other machines will finish (m_i -1) p_{imin} / p_{imax} many jobs on average. Thus equation (13) gives an estimate on the average (or steady state) number of job positions for the machine that processes jobs of the shortest processing time.

Take Fig. 2 as an example. Suppose there are five machines and two recipes for a tool group i, where eight jobs that need long time to process belong to recipe 1, and eight jobs that need short time to process belong to recipe 2. Furthermore, suppose the time took for processing a job of recipe 1 is about four times to the time for processing a job of recipe 2. Thus m_i = 5, n_i = 8+8 = 16, and p_{imin} / p_{imax} = 4. Therefore equation (13) gives Np =8, which means it suffices to assign at most 8, instead of 16, positions for scheduling a job in a machine for this tool group.

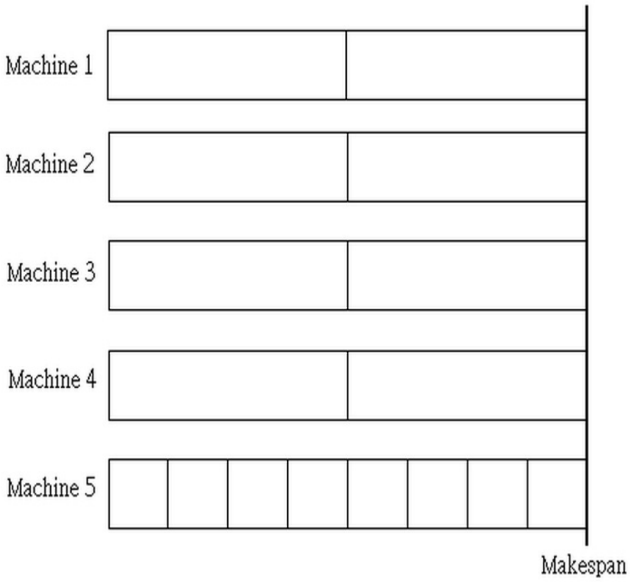

Fig. 2. A tool group example to illustrate our proposed heuristic to reduce the number of positions for job assignment in a single machine.

With this heuristic, we can successfully reduce the size of MIP, which helps to solve larger cases in shorter time. However, later in Section 5, we will show that solving our reduced MIP formulation is still too time-consuming, and not so applicable for some real-world applications which usually require a solution of good quality within short time. To this end, we propose new dispatching rule and reoptimization techniques in next section.

IV. HEURISTICS OF DISPATCHING RULES AND REOPTIMIZATION

A. A dispatching Rule Considering Least Changeover Time

In practice, a real-time on-line scheduling is too difficult to calculate. Therefore most applications conduct periodical rescheduling to renew their scheduling decisions say, every 10 minutes. In order to get a good solution quickly, we propose a new dispatching rule named EDDLC (Earliest due date with least changeovers) for this problem. EDDLC is based on the following two intuitions: (1) try to assign a job to a machine without recipe changeovers, unless necessary; and (2) if a job is going to miss its due date, assign it to the first available machine capable of processing its recipe. Fig. 3 illustrates the flow chart of EDDLC. Details of the job selection mechanism are presented in Fig. 4.

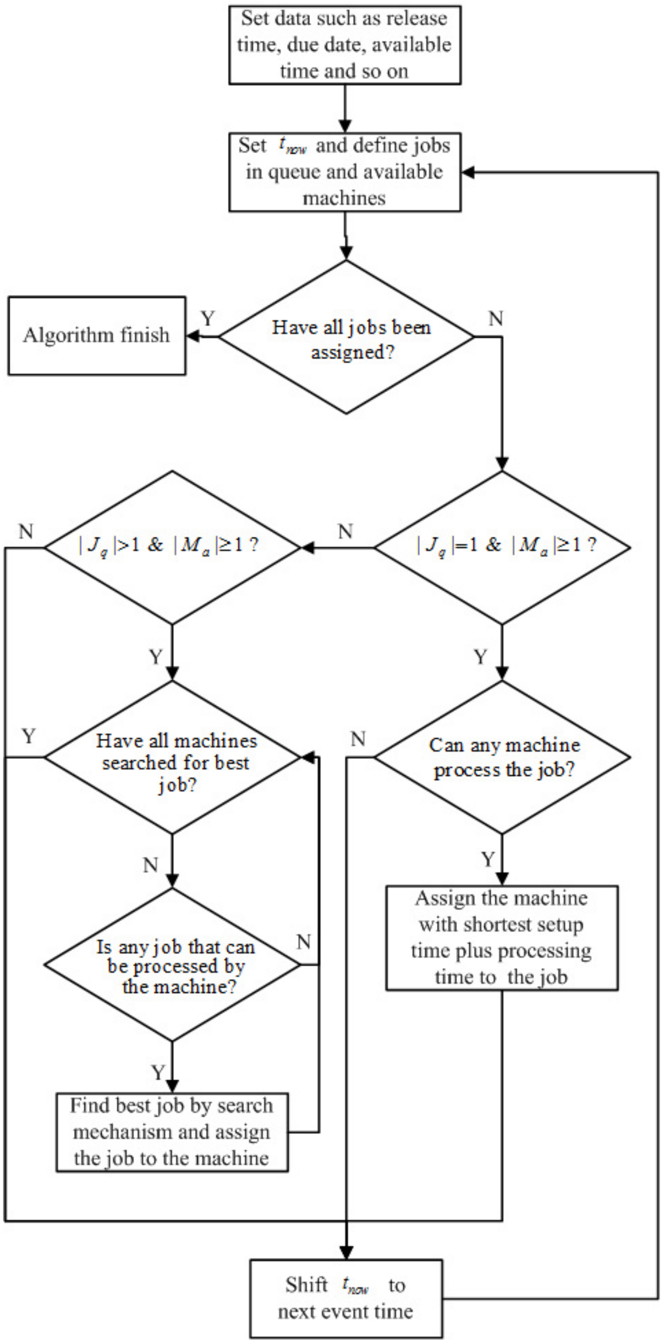

Fig. 3. Flow chart of EDDLC.

In procedures of EDDLC, different job-machine assignment mechanisms will be used, depending on the number of jobs in queue and the number of available machines at each decision point. In particular, when there is only one job in queue, EDDLC selects the machine that can process this process with the earliest finishing time. When there are more than one job in queue and more than one available machine, for each machine m_σ that is processing recipe r_δ, EDDLC first checks whether some jobs in queue are going to miss their due dates or not by equation (14). If equation (14) is satisfied for doing job j_α in m_σ, then put this job in a list according to its recipe. After all jobs have been checked for machine m_σ, EDDLC will select the job with the earliest due date from the recipe that contains the most jobs satisfying equation (14). The intuition is that EDDLC tries to reduce the number of urgent jobs for the recipe that contains the largest number of urgent jobs. On the other hand, if all jobs do not satisfy equation (14), which means no jobs are so urgent, and machine m_σ has no need to change its current recipe. In this case, machine m_σ will select the job with the earliest due date of its current recipe r_δ to process, if such a job exists. Otherwise, machine m_σ will select the job with the earliest due date of the recipe r_γ that can be processed on machine m_σ and has the shortest setup time for recipe change. If, all the jobs in queue cannot be processed on machine m_σ, then EDDLC will skip machine m_σ and continue to conduct these procedures for the next available machine.

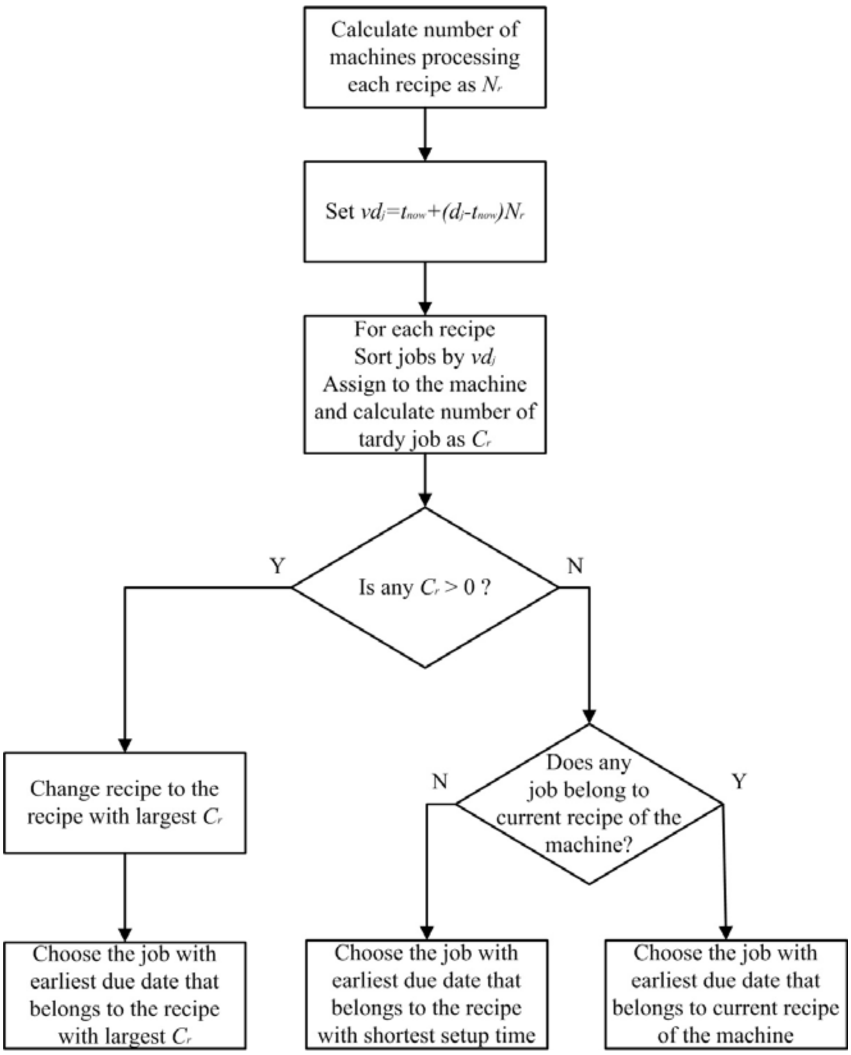

Fig. 4. Flow chart of the EDDLC job selection mechanism.

In the job selection mechanism equation (14), we use $p_{\max}$ to represent the longest processing time of jobs that belong to the recipe, denote the processing time on the machine by p, setup time for the recipe on the machine by s, the number of machines processing the recipe by N_r and due date of the ith job by d_i where jobs are sorted by due date. The intuition of equation (14) is to put the next job that can be processed in this machine into consideration, so that when a very urgent job arrives (so-called "hot job") with the worst case processing time $p_{\max}$, this machine still has room to process it without violating its due date. The third term of equation (14) in some sense represents the average time to process job i.

If the inequality of equation (14) is satisfied for some job i, it means that job is urgent (i.e. more likely to be overdue) and thus has higher priority to be processed earlier, even if a recipe changeover is necessary. Note that equation (14) focuses more on tardy jobs, rather than the makespan.

$$t_{now} + p_{\max} + \frac{s + p \times i}{N_r} \geq d_i \tag{14}$$

Equation (14) implies that when the number of machines to process current recipe (N_r) decreases, due date (d_i) decreases, or setup time (s) increases, there will be more recipe changeovers.

Here we use an example to illustrate why EDDLC may give a better result than EDD. Suppose at time 0 there are three jobs in queue with due dates 21, 46, 47 and recipe 1,2,1, respectively. Suppose there is only one machine that currently processes recipe 1. Suppose each job has processing time 10 and setup time 5. If EDD rule conducted, we will process job 1, 2, and 3 in order. As a result, their finishing time becomes 10, 25, and 40, respectively. On the other hand, EDDLC will process job 1, 3 and 2 in order with finishing times as 10, 35 and 20, respectively. In this example, the makespan by EDDLC is earlier than which by EDD. More importantly, if a hot job (denoted as job 4) with recipe 1 arrives at time 19, the finishing times for jobs 1, 2, 3 and 4 by EDD become 10, 25, 50 and 40, yet which by EDDLC become 10, 45, 20 and 30. Moreover, EDD gives a tardy job, while EDDLC give no tardy job. In general, EDDLC tends to give better result for hot jobs since equation (14) already takes $p_{\max}$ into consideration which can absorb the instant request of processing time caused by hot job.

B. Local Search Techniques

Since our objective tries to avoid tardy jobs (first priority) and then reduce the makespan as much as possible, EDDLC does serve its purpose by trying to attain similar goal. To further improve the scheduled result, we propose three local search mechanisms to reoptimize the schedule by EDDLC. Fig. 5, 6 and 7 illustrate that how these reoptimization mechanisms work. The following analyses assume each machine has $O(n)$ jobs to be processed.

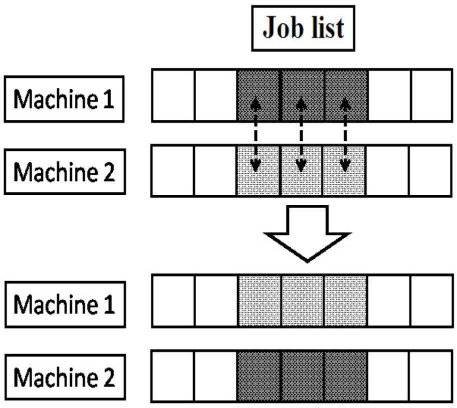

Fig. 5. The proposed interchange mechanism.

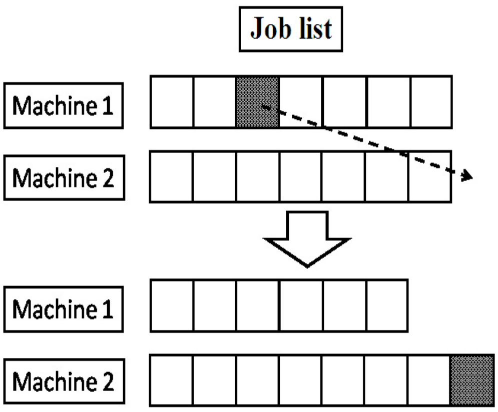

Fig. 6. The proposed translocation mechanism.

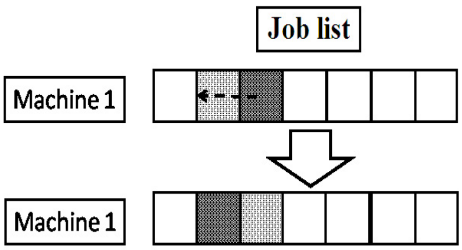

Fig. 7. The proposed transposition mechanism.

Our first mechanism, named "interchange", checks whether interchanging any two subsequences of scheduled jobs for any two machines improves the objectives, such as the number of tardy jobs, makespan and average finishing time of jobs processed by each machine lastly. In particular, for any selected two machines m_σ and m_τ, we select some consecutive jobs respectively (e.g. jobs $j_\alpha, j_{\alpha+1}, ..., j_\beta$ on m_σ and jobs $j_\eta, j_{\eta+1}, ..., j_\xi$ on m_τ) and exchange them (i.e. jobs $j_\eta, j_{\eta+1}, ..., j_\xi$ to the positions of $j_\alpha, j_{\alpha+1}, ..., j_\beta$ on m_σ, and jobs $j_\alpha, j_{\alpha+1}, ..., j_\beta$ to the positions of $j_\eta, j_{\eta+1}, ..., j_\xi$ on m_τ) to see whether such an interchange improves our objective. Since each machine has $O(n)$ jobs to be processed, each interchange takes $O(n)$ time to exchange jobs and estimate the resultant objective values. If we conduct a complete interchange for any possible subsequence between two specific machines, there will be at most $C_2^n \times C_2^n = O(n^2) \times O(n^2) = O(n^4)$ interchanges that take an overall O(n^5) time. Therefore, conducting complete interchanges for all possible two of m machines takes

$C_2^m \times O(n^5) = O(m^2 n^5)$ time.

Our second mechanism, named "translocation", checks whether inserting one job from one machine to somewhere in another machine improves the objectives. In particular, we select a job j_α from machine m_σ, and then check whether inserting it before or after the position of another job j_β in another machine m_τ helps improve the objectives. If yes, then we do the translocation; otherwise, check for another job, position, or machine. Each insertion takes $O(n)$ time to estimate its resultant objective values. If we conduct a complete translocation for any possible job pair between two specific machines, there will be at most $C_1^n \times C_1^n = O(n) \times O(n) = O(n^2)$ translocations that take an overall $O(n^3)$ time. Therefore, conducting complete translocations for all possible two of m machines takes $C_2^m \times O(n^3) = O(m^2 n^3)$ time.

Our third mechanism, named "transposition", checks whether shifting the position of a scheduled job within the same machine helps reduce the objectives. In particular, we select a job j_α from machine m_σ, and then check whether inserting it before or after the position of another job j_β in the same machine helps improve the objectives. If yes, then we do the transposition; otherwise, check for another job or position. Each insertion takes $O(n)$ time to estimate its resultant objective values. If we conduct a complete transposition for any possible job pair in a specific machine, there will be at most $C_1^n \times C_1^n = O(n) \times O(n) = O(n^2)$ transpositions that take an overall $O(n^3)$ time. Therefore, conducting complete translocations for all possible machines takes $C_1^m \times O(n^3) = O(mn^3)$ time.

C. A Proposed Heuristic Algorithm: EffROP

To integrate our proposed methodologies, we describe the steps of EffROP, our proposed scheduling algorithm, as follows:

***Algorithm* EffROP:**

Step 1: Conduct EDDLC dispatching rule.

Step 2: Repeat conducting interchange mechanism for the critical machine to each other machine, until no further improvement.

Step 3: Repeat conducting interchange mechanism for each machine to each other machine, until no further improvement.

Step 4: Conduct translocation mechanism for each machine to each other machine.

Step 5: Conduct transposition mechanism for each machine.

Step 6: If any change occurs in Step 3~5 then

Repeat conducting interchange mechanism for the critical machine to each other machine, until no further improvement.

Step 7: If any change occurs in Step 3~6 then GOTO Step 3

Otherwise, STOP.

After conducting EDDLC, algorithm EffROP iteratively applies those three proposed mechanisms to reoptimize the schedule, until no further improvement is detected. Although those repeating procedures may seem time-consuming, our computational experiments show that EffROP can in fact, terminate very fast to a solution of good quality within short time.

The complexity of EffROP can not be exactly estimated, since we can not bound the number of improvements in the process. For example, the same interchange (translocation, or transposition) operation that involves the same jobs, positions and machines may be conducted again in the process, since the initial condition keeps changing as long as the objective improves. However, we can estimate the complexity for each improvement made by one iteration of interchange, translocation or transposition, as discussed in Subsection IV B.

Among these three proposed reoptimization mechanisms, the most time-consuming mechanism is the interchange. We create 5 cases of 160 jobs and 32 machines to test and record the proportions for different number of job subsequences that have effective interchanges (i.e. interchanges of improved objective), as shown in Fig. 8. From Fig. 8, we observe that long job subsequences (e.g. the subsequence of more than 6 consecutive jobs) are almost ineffective, even if they consume most of the computational time by interchange. On the other hand, subsequences of 1 to 3 consecutive jobs contribute more than 95% of the improvement created by all possible interchanges. In other words, if we restrict the number of consecutive jobs (e.g. 1~3) to be interchanged, we save much computational time while obtaining an objective that may be very close to the complete interchange mechanism. To this end, we propose a so-called "position limit" technique that only interchanges short job subsequences (i.e. of length less than or equal to a constant k). In this case, an interchange with position limit (say, k=3) for two specific machines requires at most $C_1^n \times C_1^n = O(n^2)$ exchanges. Since each exchange takes $O(n)$ time to estimate its affect and a complete interchange checks $C_1^m \times C_1^m = O(m^2)$ possible machine pairs, a complete interchange with position limit for all possible machine pairs takes an overall $O(m^2 n^3)$ time. Our computational tests in the next section also test the effects of position limit technique, when it is used in conjunction with the EffROP and EDD dispatching rules.

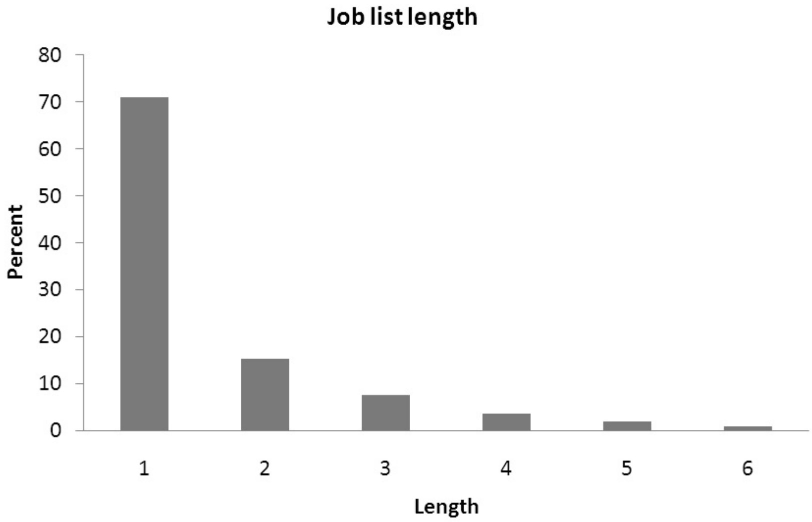

Fig. 8. Proportion of successful interchange length.

D. Tabu Search

In literature, Logendran *et al.* [18] solved similar scheduling problems by several tabu search mechanisms and showed that some of them are good with statistically significant difference for both objective and running time. To compare the efficiency and effectiveness of EffROP with tabu search, we have modified one of the tabu search mechanisms with statistically significant difference in [18] and compare it with EffROP. The tabu search continuously applies interchange and insertion to find all possible neighbor solutions and selects the best to continue and record it into a tabu list of fixed size (TLS), until the number of iterations without improvement (IWOI) reaches a specified threshold.

V. COMPUTATIONAL RESULTS AND ANALYSES

A. Settings for Computational Experiments

To have an overview on the performance for our proposed mathematical models and heuristic methodologies, we conduct computational experiments for our MIP formulations (the complete form, named "MIP", and its reduced form, named "MIPH"), EDDLC, EffROP, EffROP with position limit, EDD [3], EDD combining our reoptimization with position limit, and tabu search [18].

The MIPs are solved by ILOG CPLEX 11.1.1. All the other heuristic algorithms (including dispatching rule, reoptimization and tabu search) are writing by C++. All the tests are conducted on an Intel machine with Inter(R) Core(TM)2 CPU 6320 @ 1.86 GHz, 3 GB of RAM, and Windows XP OS.

Test cases are created based on real-world parameters by following settings: (1) 2 tool groups; (2) 16 recipes; (3) Processing time: U(10,20); (4) Setup time: U(0,5); (5) Release time: U(0,90); (6) Due date: U(Release time, Release time +450); (7) Due time: >Release time + min{processing time}; (8) Machine available time: U(0,20); (9) Number of jobs / Number of machines = 5; (10) Number of jobs that belong to recipe *r* is created in order of *r* with distribution U(0,number of rest of the jobs); (11) Number of machines that belong to tool group *i* is created with distribution U(0,number of rest of the machines). If setting (6) produces a job with due date that is earlier than its release time plus its shortest processing time, setting (7) will be invoked to force that job to be a so-called "hot job". We create ten cases for each problem set of different sizes and calculate their averaged performance in running time and objectives. Setting (10) produces primary products and subordinate products by number of jobs that belong to different recipes. Setting (11) produces changing number of machines to simulate maintenance of machines. We test with 10 to 200 jobs (with increment of 10) by these settings.

B. Sensitivity Analysis

To know whether the efficacy and efficiency of EDDLC and our reoptimization techniques are affected by some problem settings (e.g. the average number of jobs on a machine, the ratio of processing time to setup time and length of due dates), we set up the test problems by varying several parameters, as shown in Table IV. Settings of scenario 1 are the target settings, suggested by a large semiconductor manufacturing company in Taiwan. We test with 10 to 200 jobs (with increment of 10) in scenario 1 to 4, and with 20 to 200 jobs in scenario 5 to 8, respectively.

TABLE IV. CHARACTERISTIC OF EACH SCENARIO

Scenario	**Average # jobs on a machine**	**Setup time**	**Due date**
1	5	U(0,5)	U(Release time, Release time +450)
2	5	U(0,5)	U(Release time, Release time +225)
3	5	U(0,10)	U(Release time, Release time +450)
4	5	U(0,10)	U(Release time, Release time +225)
5	10	U(0,5)	U(Release time, Release time +450)
6	10	U(0,5)	U(Release time, Release time +225)
7	10	U(0,10)	U(Release time, Release time +450)
8	10	U(0,10)	U(Release time, Release time +225)

C. Results of Computational Experiments

MIP and MIPH are only tested for problems with 10 to 50 jobs, since they can not calculate an optimal solution (sometimes even a feasible solution) for all other larger problems within 10 minutes, which is a time interval commonly used for the purpose of rescheduling in practice. Since scenarios 1, 2, 3 and 4 have similar tendencies and so do scenarios 5, 6, 7 and 8, here we use the results of scenario 1 and 5 in Fig. 9 to Fig. 14 to respectively represent the tendency for these two groups of scenarios, and put all other results in APPENDIX. Some detailed results are shown in Tables V to X.

From Fig. 9 and Fig. 12, we observe that MIP and MIPH can only deal with small cases. For cases with jobs more than 30, MIP could at most obtain a feasible solution, while MIPH might get an optimal solution for some cases, which shows our heuristic to reduce the size of MIP does help to improve the solution quality for solving larger problem within shorter time. All these exact-optimal solution methods such as MIP and MIPH become too time-consuming for cases with more than 50 jobs. On the other hand, algorithm EffROP not only can give optimal or very near optimal solutions within much shorter time for smaller cases, but also calculate good solutions for large cases.

The objective values of EDD and EDDLC are about the same, as shown in Fig. 10, 11, 13 and 14, but EDDLC has much fewer recipe changeovers than EDD as shown in Table XI, where a recipe changeover rate is defined as the number of recipe changeovers divided by the number of jobs. From the design of experiments, we find that recipe changeover rate of EDDLC is affected by setup time, number of machines and due date evidently, but the recipe changeover rate of EDD is not. The results are consistent with the design intuition of equation (14), where we take these parameters into consideration to avoid recipe changeovers while achieving similar objectives of EDD. In general, EDDLC takes more advantages for cases of more machines, longer due dates and smaller setup times.

Comparing Fig. 10, 11 (with 5 jobs for each machine on average) with Fig. 13, 14 (with 10 jobs for each machine on average), we observe that the variations in solution qualities become larger when the average number of processed jobs per machine increases. This fact may be caused by the accumulated variations of more jobs to be processed by a machine.

If we take a closer look at the performance of EDD, EDDLC and those implementations with reoptimization techniques from Fig. 9 to Fig. 14, we observe that the reoptimization techniques can effectively improve the results of dispatching rules, whether starting with the schedule by EDD or EDDLC. Futhermore, these figures also indicate that our proposed position limit technique does reduce a lot of running time for EffROP while achieving similar solution qualities of EffROP.

When solving for small cases in scenario 1, tabu search and EffROP take similar amount of running time, as shown in Fig. 9. Although tabu search is a little faster than EffROP for solving cases in scenario 5, as shown in Fig. 12, it is always slower than the EffROP with position limit implementation for solving cases in both scenarios. In terms of the solution quality, Fig. 10, 11, 13 and 14 indicate the objective by tabu search is worse than all our proposed methods with reoptimization techniques such as EffROP, EffROP with position limit implementation, and EDD with reoptimization and position limit implementation. In other words, the EffROP with position limit implementation is more efficient and effective than tabu search.

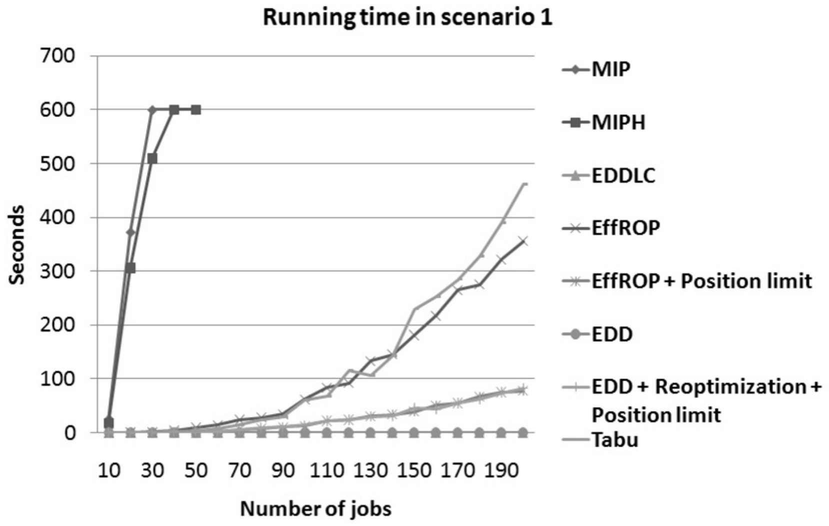

Fig. 9. Results of running time in scenario 1.

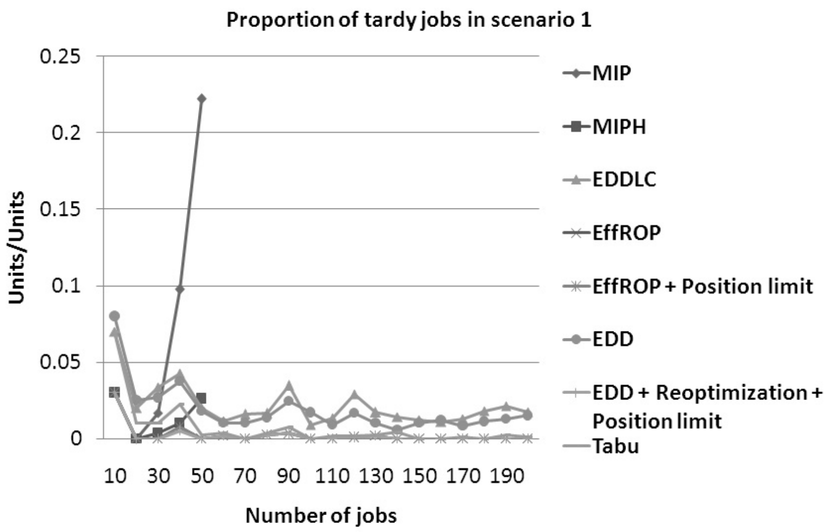

Fig. 10. Results of tardy jobs in scenario 1.

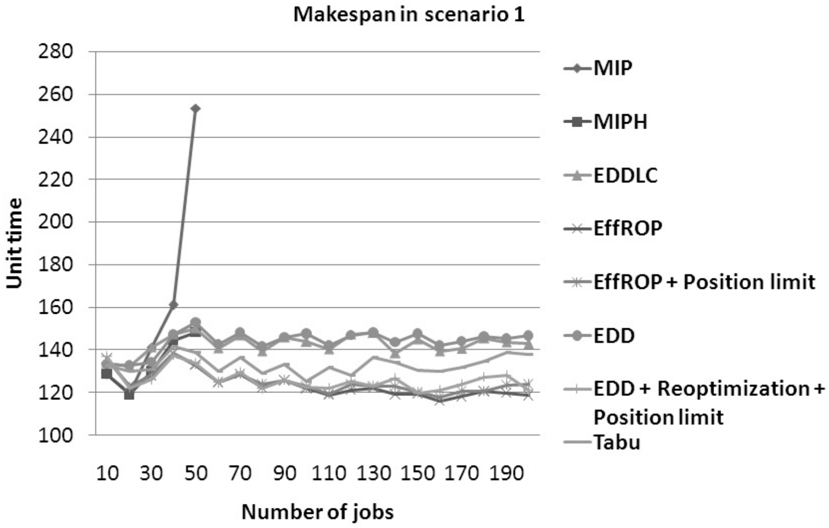

Fig. 11. Results of makespan in scenario 1.

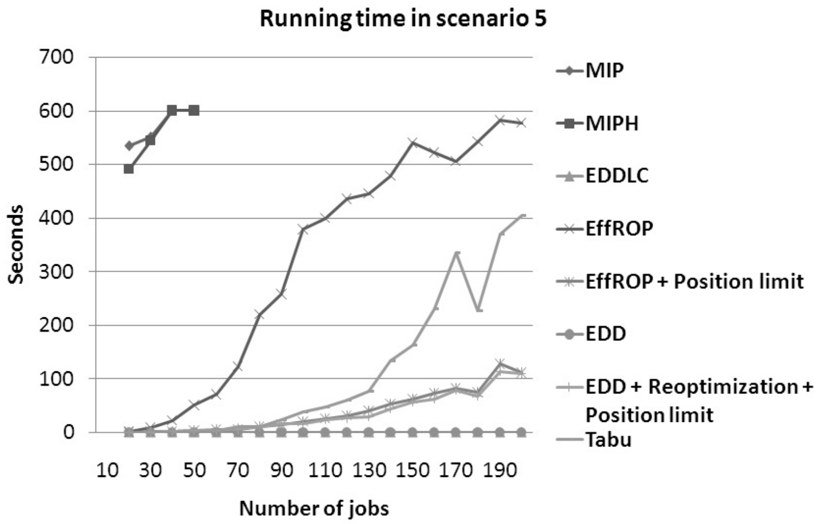

Fig. 12. Results of running time in scenario 5.

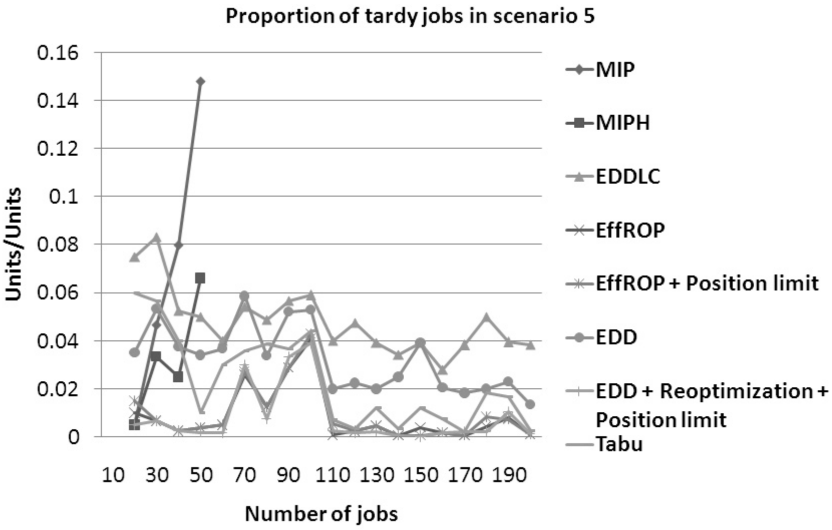

Fig. 13. Results of tardy jobs in scenario 5.

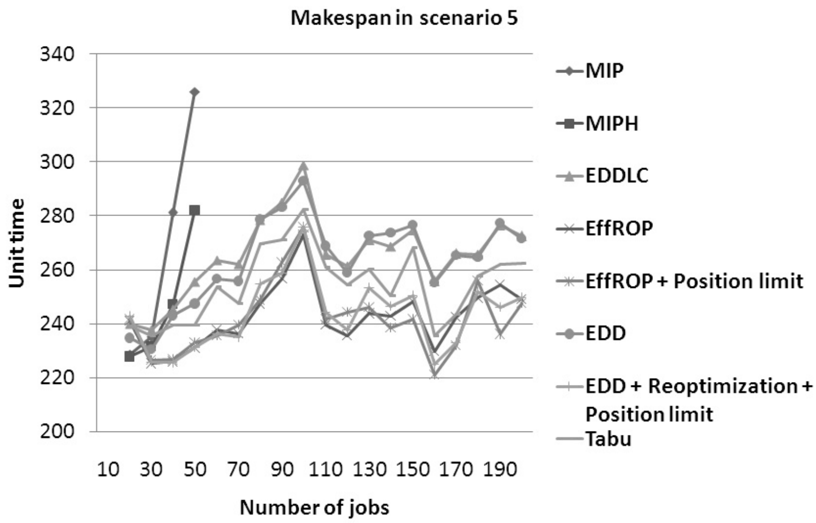

Fig. 14. Results of makespan in scenario 5.

Table V. RUNNING TIME IN SCENARIO 1.

Problem size	**40**	**80**	**120**	**160**	**200**
MIP	600.13	-	-	-	-
MIPH	600.10	-	-	-	-
EDDLC	0.00	0.02	0.04	0.08	0.14
EffROP	3.90	28.1	91.90	217.50	356.00
EffROP + Position limit	1.13	7.67	23.75	50.66	77.54
EDD	0.00	0.00	0.02	0.04	0.07
EDD + Reoptimization + Position limit	1.33	8.90	23.68	43.67	82.56
Tabu	1.24	23.01	115.23	252.05	461.69

Table VI. PROPORTION OF TARDY JOBS IN SCENARIO 1.

Problem size	**40**	**80**	**120**	**160**	**200**
MIP	0.0975	-	-	-	-
MIPH	0.01	-	-	-	-
EDDLC	0.0425	0.0162	0.0291	0.0106	0.0175
EffROP	0.0075	0.0025	0.0008	0	0
EffROP + Position limit	0.0075	0.0025	0.0008	0	0
EDD	0.0375	0.0137	0.0166	0.0118	0.015
EDD + Reoptimization + Position limit	0.005	0.0025	0.0008	0	0
Tabu	0.0225	0.0037	0.0016	0	0.001

Table VII. MAKESPAN IN SCENARIO 1.

Problem size	**40**	**80**	**120**	**160**	**200**
MIP	161.03	-	-	-	-
MIPH	144.38	-	-	-	-
EDDLC	147.31	139.15	147.31	139.38	142.83
EffROP	138.13	122.23	121.07	115.92	118.54
EffROP + Position limit	138.23	123.94	124.03	117.96	123.68
EDD	147.13	141.59	146.65	141.81	146.55
EDD + Reoptimization + Position limit	137.33	121.78	125.34	120.87	120.87
Tabu	141.60	129.02	127.98	129.81	137.69

Table VIII. RUNNING TIME IN SCENARIO 5.

Problem size	**40**	**80**	**120**	**160**	**200**
MIP	600.04	-	-	-	-
MIPH	600.06	-	-	-	-
EDDLC	0.00	0.01	0.03	0.05	0.10
EffROP	22.33	220.11	435.13	520.83	577.13
EffROP + Position limit	1.59	9.80	30.82	72.20	110.80
EDD	0.00	0.00	0.01	0.03	0.05
EDD + Reoptimization + Position limit	1.56	10.30	26.37	62.38	109.97
Tabu	0.87	11.52	61.00	229.88	405.37

Table IX. PROPORTION OF TARDY JOBS IN SCENARIO 5.

Problem size	**40**	**80**	**120**	**160**	**200**
MIP	0.08	-	-	-	-

MIPH	0.025	-	-	-	-
EDDLC	0.0525	0.0487	0.0475	0.0281	0.0385
EffROP	0.0025	0.0125	0.0025	0.0018	0.001
EffROP + Position limit	0.0025	0.0125	0.0025	0.0018	0.001
EDD	0.0375	0.0337	0.0225	0.0206	0.0135
EDD + Reoptimization + Position limit	0.0025	0.0075	0.0016	0.0018	0.0015
Tabu	0.04	0.0387	0.0033	0.0075	0.0025

Table X. MAKESPAN IN SCENARIO 5.

Problem size	**40**	**80**	**120**	**160**	**200**
MIP	281.17	-	-	-	-
MIPH	247.16	-	-	-	-
EDDLC	245.45	278.53	261.41	255.79	272.40
EffROP	225.94	247.45	235.67	229.87	248.99
EffROP + Position limit	226.70	248.86	244.20	221.04	247.54
EDD	243.08	278.58	258.80	255.19	271.65
EDD + Reoptimization + Position limit	225.59	254.79	237.68	224.92	249.78
Tabu	239.55	269.70	254.32	235.63	262.34

Table XI. RECIPE CHANGEOVER RATE.

Rules\Scenarios	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
EDD	0.4234	0.4213	0.4204	0.4217	0.3980	0.3947	0.3992	0.3990
EDDLC	0.3843	0.3842	0.4180	0.4188	0.4285	0.4521	0.4498	0.5146

D. Computational Testings on Cases of Long Job Lists

According to complexity analysis and sensitivity analysis, we know that EffROP may perform worse for cases of long job lists. Nevertheless, our proposed "position limit" technique can avoid many ineffective reoptimization operations. To verify the effectiveness of introducing the position limit technique, we compare "MIPH", "EffROP", "EffROP + position limit" and "tabu search" for 30 random cases composed by (1) 10 random cases of 100 jobs and 2 machines; (2) 10 random cases of 150 jobs and 3 machines; and (3) 10 random cases of 200 jobs and 4 machines. Note that MIP or MIPH cannot give a feasible solution within 10 minutes for all of these cases. The test results shown in Fig. 15, 16 and 17 indicate that our proposed dispatching rule and reoptimization techniques (i.e. EffROP + position limit) is the most efficient and effective solution method for cases of long job lists.

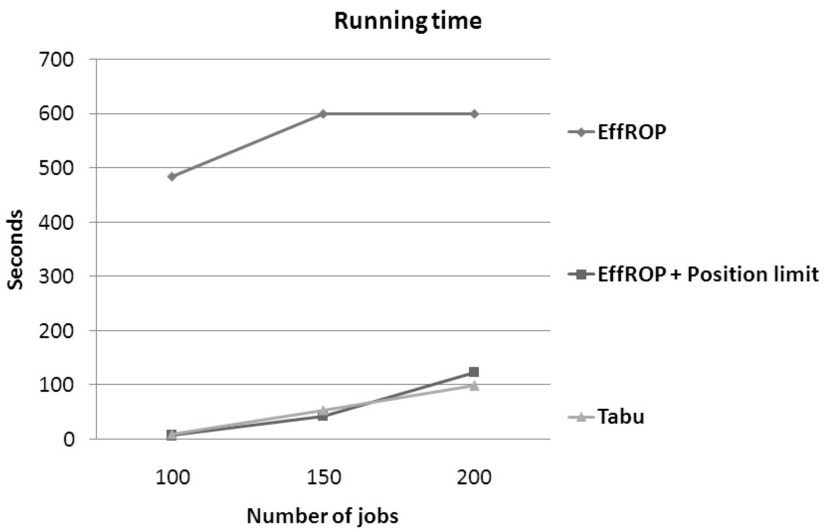

Fig. 15. Running time with long job lists.

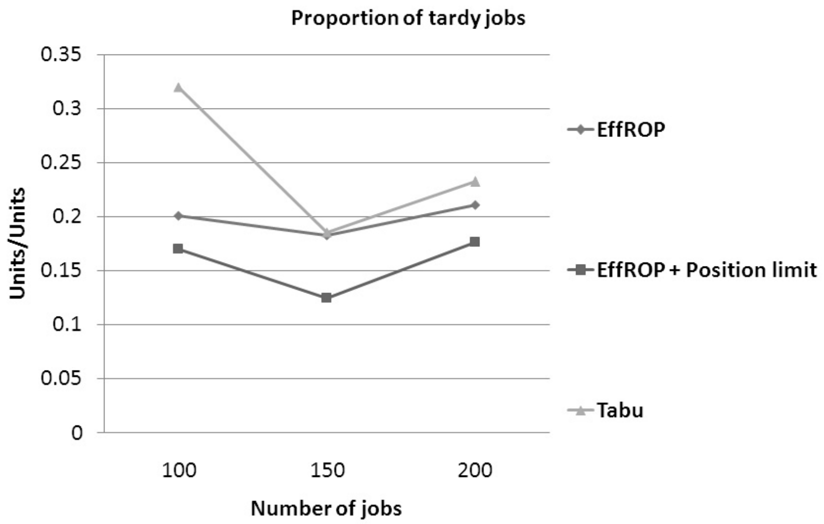

Fig. 16. Proportion of tardy jobs with long job lists.

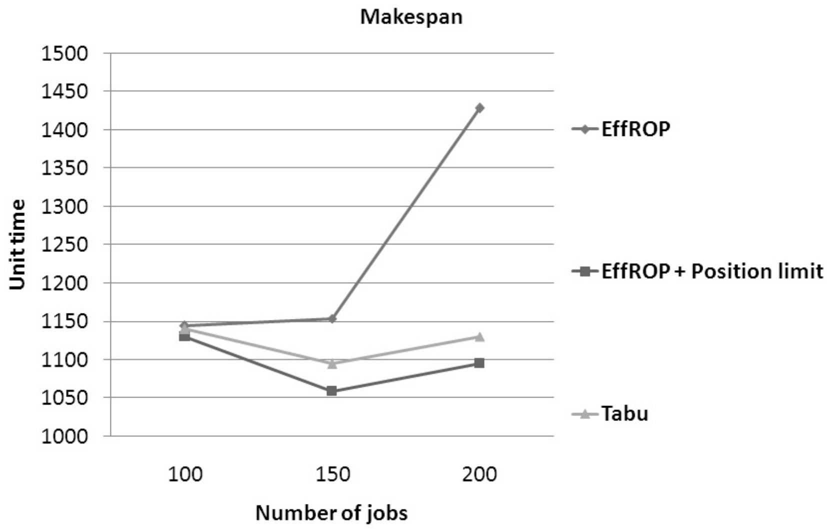

Fig. 17. Makespan with long job lists.

VI. CONCLUSIONS

A. Summary

This paper deals with a difficult scheduling problem commonly appears in semiconductor manufacturing process, which involves sequence dependent setup time, release time, due date and tool constraints. Due to the short product lifetime and competitive markets, timing to enter a target market and capability to supply sufficient products within minimum lead time whenever requested are crucial in the semiconductor manufacturing industry. A semiconductor manufacturing company that can produce more products within shorter time would dominate the market more easily. On the other hand, one that gives inefficient manufacturing plan may have to lower down their sales price for their products. Thus we investigate how to schedule all jobs to minimize number of tardy jobs and then minimize the makespan.

According to literature, the scheduling problems investigated in this paper is NP-hard, and most previous literatures focus on dispatching rules and heuristic algorithms such as tabu search and genetic algorithms. We first try to calculate an optimal solution for our scheduling problems by an MIP formulation, which takes a lot of storage space and running time even for solving small cases. We then propose a reduced MIP formulation called MIPH which estimates an upper bound on the number of jobs processed by a machine to reduce the number of variables and constraints for MIP. MIPH does give better solutions within shorter time than MIP, but it is still not suitable to deal with real-world problems in semiconductor manufacturing that often asks for an updated schedule for every few minutes. To this end, we propose EffROP that includes a new dispatching rule EDDLC and three reoptimization techniques based on local search mechanisms. Our tests show that EDDLC outperforms EDD when there are many machines, small setup times and long due dates. The EffROP based algorithms are better than tabu search and give good results very efficiently and effectively. For cases of long job lists, the position limit technique does help to avoid ineffective reoptimization operations. We suggest the use of EffROP and position limit technique for solving difficult scheduling problems in semiconductor manufacturing.

B. Contributions and Suggetions for Future Research

The major works we have completed in this paper are summarized as follows:

(1) Propose an MIP formulation in Section 3.

(2) Give techniques that can reduce the size of an MIP formulation in Section 3.

(3) Develop an intuitive dispatching rule (EDDLC) which achieve similar solution quality as EDD but with fewer recipe changeovers in Section 4.

(4) Propose three reoptimization techniques involving interchange, translocation and transposition for jobs within or between machines in Section 4.

(5) Propose an efficient and effective algorithm EffROP that integrates EDDLC and three reoptimization techniques in Section 4.

(6) Compare EDDLC with EDD in Section 5.

(7) Compare EffROP with tabu search in Section 5.

We suggest the following topics of research potential for future studies:

(1) Develop new dispatching rules for specific problems. Dispatching rules are the most commonly used techniques to deal with real-world scheduling problems in practice. Here in our problem we develop techniques to reduce recipe changeovers. There is still much room for developing better dispatching rules for other challenging scheduling problems which involve batch processes, job splitting or combining operations.

(2) Derive more precise upper bound on the number of jobs processed by a machine. In the MIPH formulation, we give an estimator for the upper bound of the number of jobs processed by a machine to reduce the size of the MIP formulation. How to identify better estimators for such an upper bound will be an interesting and challenging research topic.

(3) Better interchange scheme. According to our computational results, our proposed local search mechanisms are indeed efficient and effective to deal with this complex problem, and the proposed "position limit" technique of fixed length (e.g. 3 neighbor jobs) also effectively reduce the running time without sacrificing the solution qualities. It would be interesting to seek more systematic approaches to decide a set of consecutive jobs of dynamic size to interchange.

REFERENCES

[1] X. Zhu and W. E. Wilhelm, "Scheduling and Lot Sizing with Sequence-Dependent Setup: A Literature Review," *IIE transactions*, vol. 38, no. 11, pp. 987-1007, 2006.

[2] M. Pinedo, *Scheduling Theory Algorithm and Systems*, New Jersey: Prentice Hall Englewood Cliffs, 1995.

[3] J. H. Blackstone, D. T. Phillips, and G. L. Hogg, "A State-of-the-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations," *International Journal of Production Research*, vol. 20, no. 1, pp. 27-45, 1982.

[4] R. Haupt, "A Survey of Priority Rule-Based Scheduling," *OR Spectrum*, vol. 11, no. 1, pp. 3-16, 1999.

[5] C. Wang, , H. Ghenniwa, and W. Shen. "Heuristic Scheduling Algorithm for Flexible Manufacturing Systems with Partially Overlapping Machine Capabilities," in *Mechatronics and Automation, 2005 IEEE International Conference*, vol. 3, pp 1139-1144.

[6] Y. H. Lee, K. Bhaskaran, and M. Pinedo. "A Heuristic to Minimize the Total Weighted Tardiness with Sequence Dependent Setups," *IIE Transactions*, vol. 29, pp. 45-52, 1997.

[7] K. Altendorfer, B. Kabelka, and W. Stöher. "A New Dispatching Rule for Optimizing Machine Utilization at a Semiconductor Test Field," in *Advanced Semiconductor Manufacturing Conference, 2007, IEEE/SEMI*, pp. 188-193.

[8] I. M. Ovacik, and R. Uzsoy. "Rolling Horizon Procedures for Dynamic Parallel Machine Scheduling with Sequence-Dependent Setup Times," *International Journal of Production Research*, vol. 33, no. 11, pp. 3173-3192, 1995.

[9] J. M. J. Schutten, "List Scheduling Revisited," *Operations Research Letters*, vol. 18, no. 4, pp. 167-170, 1996.

[10] M. E. Kurz, and R. G. Askin. "Heuristic Scheduling of Parallel Machines with Sequence-Dependent Set-up Times," *International Journal of Production Research*, vol. 39, no. 16, pp. 3747-3769, 2001.

[11] S. S. Kim, H. J. Shin, D. H. Eom, and C. O. Kim. "A Due Date Density-Based Categorising Heuristic for Parallel Machines Scheduling," *The International Journal of Advanced Manufacturing Technology*, vol. 22, no. 9, pp. 753-760, 2003.

[12] N. Balakrishnan, J. J. Kanet, and S. V. Sridharan. "Early/Tardy Scheduling with Sequence Dependent Setups on Uniform Parallel Machines," *Computers & Operations Research*, vol. 26, no. 2, pp. 127-141, 1999.

[13] F. Sivrikaya-Serifoglu, and G. Ulusoy. "Parallel Machine Scheduling with Earliness and Tardiness Penalties," *Computers & Operations Research*, vol. 26, no. 8, pp. 773-787, 1999.

[14] Ü. Bilge, F. Kırac M. Kurtulan, and P. Pekgün. "A Tabu Search Algorithm for Parallel Machine Total Tardiness Problem," *Computers & Operations Research*, vol. 31, no. 3, pp. 397-414, 2004.

[15] Y. Arzi, and D. Raviv. "Dispatching in a Workstation Belonging to a Re-Entrant Production Line under Sequence-Dependent Set-up Times," *Production Planning & Control*, vol. 9, no. 7, pp. 690-699, 1998.

[16] J. Bank, and F. Werner. "Heuristic Algorithms for Unrelated Parallel-Machine Scheduling with a Common Due Date, Release Dates, and Linear Earliness and Tardiness Penalties," *Mathematical and Computer Modelling*, vol. 33, pp. 363-383, 2001.

[17] C. O. Kim, and H. J. Shin. "Scheduling Jobs on Parallel Machines: A Restricted Tabu Search Approach," *The International Journal of Advanced Manufacturing Technology*, vol. 22, no. 3, pp. 278-287, 2003.

[18] R. Logendran, B. McDonell, and B. Smucker. "Scheduling Unrelated Parallel Machines with Sequence-Dependent Setups," *Computers & Operations Research*, vol. 34, no. 11, pp. 3420-3438, 2007.