

Scheduling with Advanced Process Control Constraints

Yiwei Cai*, Erhan Kutanoglu†, John Hasenbein‡, Joe Qin§

July 2, 2009

Abstract

With increasing worldwide competition, high technology manufacturing companies have to pay great attention to lower their production costs and guarantee high quality at the same time. Advanced process control (APC) is widely used in semiconductor manufacturing to adjust machine parameters so as to achieve satisfactory product quality. When there is a conflict between quality and scheduling objectives, quality usually has to be satisfied. This paper studies the interaction between scheduling and APC. A single-machine multiple-job-types makespan problem with APC constraints is proved to be NP-hard. For some special cases, optimal solutions are obtained analytically. In more general cases, the structure of optimal solutions is explored. An efficient heuristic algorithm based on these structural results is proposed and compared to an integer programming approach.

*Freescale Semiconductor, Inc., 3501 Ed Bluestein Boulevard., Austin, Texas 78721, ewaycai@gmail.com

†Graduate Program in Operations Research and Industrial Engineering, Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX, 78712, erhank@mail.utexas.edu

‡Graduate Program in Operations Research and Industrial Engineering, Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX, 78712, jhas@mail.utexas.edu

§The Mork Family Department of Chemical Engineering and Materials Science, Ming Hsieh Department of Electrical Engineering, Daniel J. Epstein Department of Industrial and Systems Engineering, University of Southern California, 925 Bloom Walk, HED 211, Los Angeles, CA 90089-1211, sqin@usc.edu

1 Introduction

The semiconductor production environment has characteristics of a high-mix of products and orders, process flows with reentrant structure, long cycle times, complex operational constraints such as time limits between operations, and job priorities. Each process flow in a fab can contain 200-500 processing steps and utilize more than one hundred machines, each costing as much as tens of millions of dollars. Even small increases in efficiency yield enormous benefits.

With the rapid development of new technology and increasing worldwide competition, high technology manufacturing companies have to pay more attention to lower their production costs and guarantee high quality at the same time. Short production cycle times and high product quality are very desirable for semiconductor manufacturing companies.

Advanced Process Control (APC) is a broad term that encompasses different process control tools such as model predictive control (MPC), process modeling, system identification, and controller performance monitoring and diagnosis. APC is widely used in chemical industries to improve product quality or reduce operation cost. In semiconductor manufacturing, APC determines control parameters for machines based on measurement results so that the quality of the wafer is guaranteed [2, 3, 6].

Production scheduling has been studied extensively for decades. Recently, more semiconductor manufacturing companies have observed that APC requirements provide additional constraints which make the scheduling problem even more complicated. Scheduling decisions in turn affect the APC results. Therefore it is critical to investigate the relation between these two components. How to make better decisions for both scheduling and APC is a timely topic for both academic researchers and industry practitioners.

This paper focuses on one aspect of the interplay between APC and scheduling in semiconductor manufacturing systems. In a semiconductor fab, a Kalman filter based approach can be used to control multi-product/-process systems [5]. Once a job is completed, the state of the completed job is used in an APC model to predict the state of the next job. Based on the prediction, control parameters are determined for the next operation to achieve certain predefined set points. In a normal APC model, the level of uncertainty in individual state estimates is calculated from the trace of a time-varying error covariance matrix. In semiconductor manufacturing, large errors are mainly caused by process drift as time goes on [6, 7]. The size of the errors in the states is affected by the order in which jobs are processed and measured. Therefore, in order to obtain a good state prediction and thus achieve higher product quality, the APC model keeps updating the sta-

tus of different layer/machine combinations. If a machine has not processed a specific layer of a wafer type for some time, the APC model may require a so-called “qualification-run” (or qual-run for short) procedure to reduce the prediction error and make sure the machine is capable of processing this specific layer.

In a qual-run procedure, a blank wafer is processed and measured to obtain the status of the machine so that the APC model can determine proper levels for the machine parameters that will achieve high quality results for specific jobs. Before the result of the qual-run is available, the job cannot be processed on the machine, which results in the loss of productive machine capacity. Hence, the coordination between scheduling, which tries to keep the machines utilized with important jobs, and APC, which tries to keep the APC model up to date, is critical for semiconductor manufacturing. This paper considers a single-stage scheduling problem with APC (mainly qual-run) constraints to obtain insight into this coordination issue.

Since the APC problem involves qual-runs and setups, it is necessary to briefly review some previous work related to setup scheduling. There are many studies regarding production scheduling with setups. Monma and Potts [4] extended various one-machine scheduling models, such as those which seek to minimize maximum lateness or total weighted completion time, to include batch setup time. They proposed a dynamic programming approach which results in an algorithm that is polynomially bounded in the number of jobs, but is exponential in the number of batches. Webster and Baker [10] reviewed the literature on single-machine scheduling models that incorporate benefits from job grouping. Zdrzalka [13] studied a single-machine scheduling problem with release and delivery times, and sequence-independent setups between any two different families. A branch and bound algorithm is developed to solve problem instances with hundreds of jobs quickly. Yang and Liao [11] did an extensive survey of static scheduling research involving setup times. In their paper, the literature is classified into job, class, and job-and-class setup situations. Yuan et al. [12] considered the single machine batch scheduling problem with family setup times and release dates to minimize the makespan. They showed that this problem is strongly NP-hard, and provided two dynamic programming algorithms. Uzsoy and Velásquez [9] studied the scheduling of a single machine with family-dependent setup times. In order to minimize maximum lateness, they implemented a rolling horizon heuristic and an incomplete dynamic programming heuristic that compares partial schedules and retains the better one.

The traditional setup scheduling problems mainly focus on how to reduce setups, e.g., how to process as many jobs as possible from the same family once a setup occurs. For some scenarios in the APC scheduling problem,

however, it may be beneficial to have more instead of fewer setups in order to avoid a qual-run. This paper focuses on identifying the best balance between setups and qual-runs, and thus is very different from the traditional setup scheduling problems. Although there are hundreds of studies concerning scheduling problems with setups, to the best of our knowledge, our research is the first work which attempts to coordinate scheduling and APC decisions explicitly.

This paper is organized as follows. Section 2 defines the APC scheduling problem and introduces the notation. Section 3 provides an integer programming (IP) model with a goal programming approach to solve this problem. In Section 4, two versions of the problem are proved to be strongly NP-hard. Section 5 explores the properties of the optimal solutions in special cases. Section 6 takes the advantage of the analysis in Section 5 to propose a heuristic algorithm. Section 7 compares the heuristic algorithm and the IP model. Section 8 concludes the paper with our observations and results.

2 Problem Definition

This paper considers a single-machine multiple-family scenario, and our analysis attempts to coordinate the the scheduling of setups and the “qual-run” requirements from APC. As mentioned in Section 1, APC achieves the smallest estimation error through schedules which frequently change between job families. If changes are frequent enough, then it is possible that no qual-runs are needed. However, frequent changeovers generally reduce scheduling performance and cause loss of capacity due to excessive setups.

To achieve short cycle times and high product quality simultaneously, both APC and scheduling need to sacrifice a little bit. In order to achieve the best trade off between quality and cost, APC decides a parameter range which represents an acceptable quality level for every family. Based on this range, APC provides a threshold value for every family. This threshold determines how frequently qual-runs need to be done. To simplify notation, such a threshold is called here an “*APC-value*.” In theory, there are two types of APC-values:

1. Time-based: APC-value is measured in terms of time (T_i);
2. Count-based: APC-value is measured in terms of number of jobs (n_i).

Only the count-based APC-value is widely used in practice. Thus, the majority of this paper only deals with count-based APC constraints.

After the machine finishes a job of Family i , the next time the machine processes a job from the same Family i , it needs to check if the time-based

or count-based interval between the two runs of the family exceeds the APC-value. If the interval exceeds the APC-value, a qual-run is required to obtain updated data on the machine. Before the result of the qual-run for Family i is available, no job of Family i can be processed on that machine, which results in the loss of productive machine capacity.

The following assumptions are made to facilitate discussion in the remaining part of this paper:

- We consider a single-machine problem, with a single shift over which the schedule of jobs from multiple families is decided.
- Family i is given a processing target m_i , which means the target is to finish m_i jobs of Family i in one shift. The processing target is typically provided by management or as the output of a planning model.
- Only current WIP (Work-In-Progress) inventory is considered, and there are no more incoming jobs in the shift.
- The setup time for changeovers between jobs from different families is sequence-independent, and no setups are performed between jobs from the same family.
- A setup is executed before every qual-run.
- The very first job, no matter to which family it belongs, in the schedule does not need a qual-run.

The two main objectives are (1) to minimize the deviation from the production targets across all families, and (2) minimize the total time spent for setups and qual-runs. Below, the relevant notation is introduced. The models and solution methods in the paper also can address the case where one is only interested in minimizing setups and qual-runs.

Sets:

I set of product families, indexed by i

J set of positions in the schedule, indexed by j . If one job of Family i is the j th job to be processed in the whole schedule, then the position of that job is denoted by j .

Input Parameters:

p_i processing time for jobs of Family i

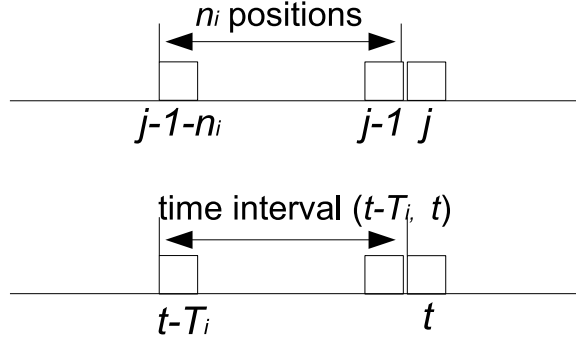


Figure 1: Count-based and time-based APC-value

q_i qual-run time for jobs of Family i

s_i setup time for jobs of Family i

n_i count-based threshold value given by APC. If Family i is processed at position j , and a job of Family i has not been processed from position $(j - 1 - n_i)$ to $j - 1$, then a qual-run is required for Family i before the job is processed at position j (see Figure 1)

T_i time-based threshold value given by APC. If Family i is processed at time t , and a job of Family i has not been processed within the time interval $(t - T_i, t)$, then a qual-run is required for Family i before the job is processed at time t (see Figure 1)

m_i processing target for Family i (this is the number of jobs of Family i to be processed during the shift)

u_i initially available WIP level for Family i

c machine capacity, i.e., the available processing time during the shift.

3 Integer Programming Formulation

This section presents an integer programming model to solve the problem incorporating setups and qual-runs, and other scheduling issues on a single machine. Since the APC problem is proved to be strongly NP-hard in Section 4, the integer programming (IP) model only serves as a benchmark to

evaluate the proposed heuristic algorithm in Section 6. This model determines the optimal processing sequence for the current WIP while trying to achieve a balance between two major objectives: minimization of the deviation from processing targets across all families, and minimization of the total time spent for setups and qual-runs. In industry, it is generally more important to meet the processing targets than to minimize the setup and qual-run time, so a goal programming approach is used to first minimize the deviation from the processing targets without explicit consideration of setup and qual-run times. Then, a similar problem is solved to minimize the total setup and qual-run time. Alternatively, it is possible to use one objective function to minimize both the processing targets and the setup and qual-run time. However, it is not easy to specify the weights between these two objectives, and different weights may obscure the observations. Thus, we use the goal programming approach in this paper. The following decision variables are used:

$X_{i,j}$ binary variable, 1 if a setup is required for a job of Family i processed at the j th position; 0 otherwise

$Y_{i,j}$ binary variable, 1 if Family i is processed at the j th position; 0 otherwise

$Z_{i,j}$ binary variable, 1 if a qual-run is required for a job of Family i processed at the j th position; 0 otherwise.

Obviously, from the point of view of our optimization model, there is no reason to produce more than the processing targets. Therefore, minimizing the deviation from the processing targets is the same as maximizing the number of jobs processed under the constraint that we do not exceed the processing targets. The proof is similar to that of Proposition 1 in [8]. Due to the same reason, without loss of generality u_i is set to be equal to m_i in the remainder of this paper. The model of achieving the first goal (minimizing deviations from processing targets) can be written as an IP as follows:

$$\max \sum_{i \in I} \sum_{j \in J} Y_{i,j}$$

subject to

$$\sum_{j \in J} Y_{i,j} \leq m_i \quad \forall i \in I \quad (1)$$

$$Y_{i,j} - Y_{i,j-1} \leq X_{i,j} \quad \forall i \in I, \forall j \in J \quad (2)$$

$$Y_{i,j} - \sum_{j'=j-n_i}^{j-1} Y_{i,j'} \leq Z_{i,j} \quad \forall i \in I, \forall j \in J \quad (3)$$

$$\sum_{i \in I} \sum_{j \in J} (s_i \cdot X_{i,j} + q_i \cdot Z_{i,j} + p_i \cdot Y_{i,j}) \leq c \quad (4)$$

$$\sum_{i \in I} Y_{i,j} \leq 1 \quad \forall j \in J \quad (5)$$

$$X_{i,j}, Y_{i,j}, Z_{i,j} \in \{0, 1\} \quad \forall i \in I, \forall j \in J. \quad (6)$$

Constraint (1) guarantees that the total number of processed jobs is not more than the minimum of the processing target and the total number of available jobs for each family. Constraints (2) and (3) determine whether a setup or qual-run for jobs of Family i is necessary at position j . Constraint (4) implies that the sum of processing time, setup time and qual-run time can not exceed the capacity (available time for processing) of the machine. Constraint (5) requires that at most one job can be placed at each position. Constraint (6) lists the binary restrictions on decision variables.

Suppose the optimal solution to the above model is $Y_{i,j}^*$, and $\tilde{m}_i = \sum_{j \in J} Y_{i,j}^*$.

After the above IP model is solved, the second model is used to minimize the total time on setups and qual-runs. Using the same notation, the model is:

$$\min \sum_{i \in I} \sum_{j \in J} (s_i \cdot X_{i,j} + q_i \cdot Z_{i,j})$$

subject to

$$\sum_{j \in J} Y_{i,j} = \tilde{m}_i \quad \forall i \in I \quad (7)$$

$$Y_{i,j} - Y_{i,j-1} \leq X_{i,j} \quad \forall i \in I, \forall j \in J \quad (8)$$

$$Y_{ij} - \sum_{j'=j-n_i}^{j-1} Y_{i,j'} \leq Z_{i,j} \quad \forall i \in I, \forall j \in J \quad (9)$$

$$\sum_{i \in I} \sum_{j \in J} (s_i \cdot X_{i,j} + q_i \cdot Z_{i,j}) \leq c - \sum_{i \in I} p_i \cdot \tilde{m}_i \quad (10)$$

$$\sum_{i \in I} Y_{i,j} \leq 1 \quad \forall j \in J \quad (11)$$

$$X_{i,j}, Y_{i,j}, Z_{i,j} \in \{0, 1\} \quad \forall i \in I, \forall j \in J. \quad (12)$$

Constraint (7) ensures that the total number of processed jobs of each Family i equals \tilde{m}_i , which is calculated from the first-level IP model. Constraint (10) requires that all the setup and qual-run times are less than or equal to the available machine capacity excluding the total processing times for all jobs. Other than the objective function and these two constraints, this model is the same as the first-level one.

4 Complexity Study

This section only considers a relaxed problem in which the machine capacity is large enough to meet all production targets with any non-idling schedule. Thus, the first objective in Section 2 is already taken care of. Obviously, if the relaxed problem is proved to be strongly NP-hard, the original problem is strongly NP-hard. Since the total processing time is fixed in this case, minimizing the makespan is equivalent to minimizing the total number of setups and qual-runs. Therefore, the objective of the relaxed problem in this section is to minimize the makespan, which is equivalent to our original objectives when machine capacity is unlimited. The 3-partition problem, which is strongly NP-complete, can be reduced to the decision versions of both types of APC-value problems. Thus, both APC-value problems are NP-hard. The 3-partition problem is as follows:

3-partition: Given a multiset S of $3t$ positive integers, with the property

that $\sum_{i=1}^{3t} p_i = t \cdot b$ and $b/4 < p_i < b/2, \forall p_i \in S$, is there a partition

S_1, \dots, S_t , such that $\sum_{i \in S_j} p_i = b, \forall j = 1, \dots, t$?

Below, strong NP-hardness is first proved for the time-based problem and then for the count-based problem.

Lemma 4.1. *Suppose a single-machine scheduling problem with time-based APC constraints has $3t+1$ families: $0, 1, 2, \dots, 3t$. The number of jobs in each family is: $m_i = t + 1$. Other parameters of the problem satisfy the following conditions:*

$$T_0 = (1 + t) \cdot b + \sum_{i=1}^{3t} s_i \quad (13)$$

$$(2 \cdot T_0 + p_0) < T_i < 2 \cdot p_0 \quad \forall i \in S = \{1, \dots, 3t\} \quad (14)$$

$$q_i > (t - 1) \cdot \sum_{i=1}^{3t} s_i, \quad (15)$$

where, b is a positive integer. If $\sum_{i \in S} p_i = tb$, and $\{S_j \mid j = 1, \dots, t\}$ is a partition of $\{p_1, \dots, p_{3t}\}$ that solves the 3-partition problem such that $\sum_{i \in S_j} p_i = b$, then, a lower bound on the makespan is:

$$M^* = t \sum_{i=0}^{3t} s_i + (t + 1) \sum_{i=0}^{3t} p_i. \quad (16)$$

Also, the optimal schedule A_t^* must have the following characteristics:

T-1 A single job of Family 0 is processed exactly at time $kT_0 + (k - 1)p_0$, where $k \in \{1, \dots, t\}$. The last job of Family 0 is processed at $t(T_0 + p_0)$.

T-2 Within each interval $[(k - 1)(p_0 + T_0), T_0 + (k - 1)(p_0 + T_0)]$, where $k = 1, \dots, t$, there are two jobs from each of the three families in S_k , and exactly one job from each of all other families in $S \setminus S_k$.

Proof. See Appendix A.1. □

We are now prepared to prove the main complexity theorem for the scheduling problem with time-based APC constraints.

Theorem 4.2. *The single-machine scheduling problem with time-based APC constraints is strongly NP-hard.*

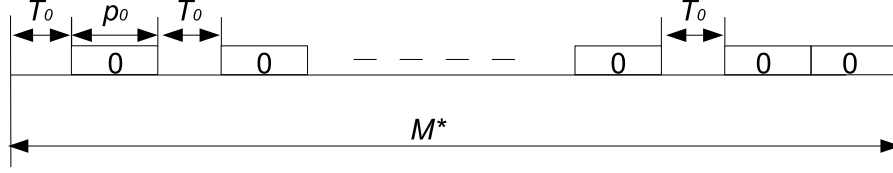


Figure 2: Strong NP-hardness for the time-based APC constraints

Proof. The decision version of the problem is proved to be strongly NP-complete. The decision problem is stated as follows: given a single-machine makespan problem with time-based APC constraints, is there a schedule whose makespan is equal to or lower than a predetermined value M^* ?

The 3-partition problem can be reduced to this problem. The instance is constructed as follows. There are $3t + 1$ families: $0, 1, 2, \dots, 3t$. The number of jobs in each family is: $m_i = t + 1$ for $\forall i \in \{0, \dots, 3t\}$. Denote the APC-value of Family i as T_i , $\forall i = 0, \dots, 3t$. The parameters satisfy the following conditions:

$$\sum_{i=1}^{3t} p_i = t \cdot b \quad (17)$$

$$b/4 < p_i < b/2 \quad \forall i \neq 0 \quad (18)$$

and conditions (13) to (15). These conditions are easy to satisfy as noted in the proof of Lemma 4.1.

If a 3-partition problem is solved, then it is possible to divide $\{p_1, \dots, p_{3t}\}$ into t sets S_k ($k = 1, \dots, t$) such that the total processing time in each group equals b . Thus, the makespan in (16) can be achieved based on Lemma 4.1. On the other hand, if a schedule with the makespan in (16) is obtained for this case, then the form of the schedule must satisfy (T-1) and (T-2) by Lemma 4.1. Then, all the additional jobs of Family i_k ($i_k \in S_k$, $k = 1, \dots, t$) in the interval $[k(p_0 + T_0), k(p_0 + T_0) + T_0]$ constitute a solution for the 3-partition problem. Therefore, the single-machine scheduling problem with time-based APC constraints is strongly NP-hard. \square

The next lemma is used to prove that the count-based problem is also strongly NP-hard.

Lemma 4.3. *Suppose a single-machine scheduling problem with count-based APC constraints has $(3t + 1)$ families: $0, 1, \dots, 3t$. The parameters satisfy the*

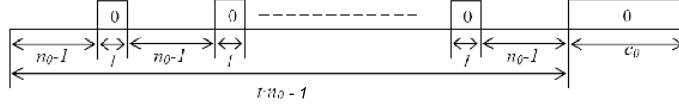


Figure 3: Strong NP-hardness for the count-based APC constraints

following conditions:

$$n_i = t \cdot n_0 - m_i \quad \forall i > 0 \quad (19)$$

$$q_i > (t - 1) \cdot s_0 \quad \forall i \geq 0 \quad (20)$$

$$s_i > (t - 1) \cdot s_0 \quad \forall i > 0. \quad (21)$$

where, n_0 is a positive integer. If $\sum_{i=1}^{3t} m_i = t(n_0 - 1)$, and $\{S_j \mid j = 1, \dots, t\}$ is a partition of $\{m_1, \dots, m_{3t}\}$ that solves the 3-partition problem such that $\sum_{i \in S_j} m_i = n_0 - 1$, then, a lower bound on the makespan is:

$$z = \sum_{i=1}^{3t} s_i + t \cdot s_0 + \sum_{i=0}^{3t} m_i \cdot p_i. \quad (22)$$

Additionally, the optimal schedule A_c^* must satisfy the following conditions (Figure 3):

C-1 One single job of Family 0 is processed at positions $k \cdot n_0$ ($k = 1, \dots, t - 1$), and $(m_0 - t + 1)$ jobs of Family 0 are processed from position $t \cdot n_0$ to position $t \cdot n_0 + (m_0 - t + 1)$.

C-2 All jobs from families other than Family 0 are processed sometime before position $t \cdot n_0$ without any partition.

Proof. See Appendix A.2. □

With this lemma in hand, we now present the main complexity result for the scheduling problem with count-based APC constraints.

Theorem 4.4. *The single-machine scheduling problem with count-based APC constraints is strongly NP-hard.*

Proof. The decision version of the problem is proved to be strongly NP-complete. The decision problem is stated as follows: given a single-machine makespan problem with count-based APC constraints, is there a schedule whose makespan is equal to or lower than a predetermined value z ?

Here is the instance of the single-machine makespan problem with count-based APC constraints. There are $(3t + 1)$ families: $0, 1, \dots, 3t$. The parameters satisfy the following conditions:

$$(n_0 - 1)/4 < m_i < (n_0 - 1)/2, \quad \forall i > 0 \quad (23)$$

$$\sum_{i=1}^{3t} m_i = t \cdot (n_0 - 1), \quad (24)$$

and conditions (19) to (21). Conditions (23) and (24) are the usual conditions for the 3-partition problem. For all $i \neq 0$, the n_i 's and q_i 's are only present in conditions (19) and (20), respectively. Thus, it is easy to find parameters which satisfy these conditions.

If a 3-partition problem is solved, then it is possible to divide $\{m_1, \dots, m_{3t}\}$ into $3t$ groups such that the total processing time in each group equals $(n_0 - 1)$. Based on these conditions and Lemma 4.3, the solution to the 3-partition problem and the schedule of Family 0 satisfying (C-1) constitute a feasible schedule whose makespan is $2t$. On the other hand, if there is a schedule S whose makespan is z , then based on Lemma 4.3, the optimal schedule must satisfy (C-1) and (C-2). Thus, the sequencing of all families other than Family 0 constitute a solution to the corresponding 3-partition problem. Therefore, the problem is strongly NP-hard. \square

In semiconductor manufacturing practice, count-based APC constraints are widely used, and thus the rest of this paper focuses on the scheduling problem with count-based APC constraints only.

5 Analysis of Special Cases

This section analyzes several special cases with the goal of using them in developing an efficient algorithm to solve more general problems. To keep things simple, in this section we consider only the problem in which there is enough machine capacity to finish all processing targets for all families. Also, in Subsections 5.1 and 5.2, a “batch” means a group of jobs of the same family which are processed consecutively. “Batch size” means the number of jobs in the group. A “block” is a group of jobs from possibly different families that are processed consecutively with no qual-run required. First, the optimal

schedule is derived for the two-family case, and then several properties of the optimal schedule for the N -family case are discussed.

5.1 Two-family Case

In this setting, only two different families exist in the system. The optimal schedule is given in Lemma 5.2. Below, $\lceil x \rceil$ denotes the smallest integer which is greater than or equal to x .

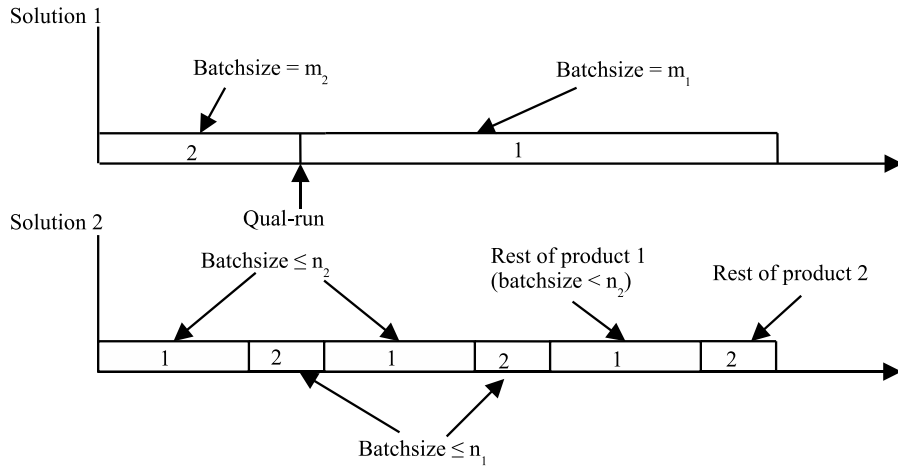


Figure 4: Two solutions in Lemma 5.1

Lemma 5.1. *Suppose $\lceil m_1/n_2 \rceil \leq \lceil m_2/n_1 \rceil$ and $s_1 = s_2 = s$. If $\min\{q_1, q_2\} < 2s(\lceil m_1/n_2 \rceil - 1)$, then a schedule composed of two batches minimizes the makespan. Otherwise, the optimal schedule processes job from Families 1 and 2 alternately. In the second case, every batch of Family 1 and Family 2 except the last batch has no more than n_2 and n_1 jobs, respectively. Furthermore, Families 1 and 2 have the same number $\lceil m_1/n_2 \rceil$ of batches, and Family 1 is processed first.*

Proof. Figure 4 provides an illustration of the optimal schedules. First, there are only two solution candidates: (1) two big batches, or (2) alternately processing jobs of Families 1 and 2 with batch sizes of no more than n_2 and n_1 , respectively. For solution (1), there is one qual-run and two setups (one at the beginning, and another after finishing the first batch). If there are more than two batches, and if there is at least one batch of Family i except the last batch with size greater than n_j ($i \neq j$), then there has to be at least one qual-run and three setups, which is worse than solution (1). Therefore,

the only solution with more than two batches that could be better than (1) is solution (2), where there is no qual-run. If every batch of Family 1 and 2 is strictly less than n_2 and n_1 , respectively, then there are more than $\lceil m_1/n_2 \rceil$ batches. Such a schedule has more setups than solution (2), hence it is suboptimal. Therefore, it is sufficient to consider only schedules which are in the form of (1) or (2).

In solution (1), the total time used for setups and qual-runs is $\min\{q_1, q_2\} + 2s$. In solution (2), in order to achieve the minimum number of setups without any qual-run, the total number of batches of Family 1 and Family 2 jobs has to be $\min\{\lceil m_1/n_2 \rceil, \lceil m_2/n_1 \rceil\}$. Since $\lceil m_1/n_2 \rceil \leq \lceil m_2/n_1 \rceil$, the total number of setups is $2\lceil m_1/n_2 \rceil$. Hence, the total setup and qual-run time is $2s\lceil m_1/n_2 \rceil$ in solution (2). If $\min\{q_1, q_2\} > 2s(\lceil m_1/n_2 \rceil - 1)$, then solution (2) is better; otherwise solution (1) is better. \square

We now relax the $s_1 = s_2 = s$ condition, and obtain the following lemma for the general case.

Lemma 5.2. *Suppose $\lceil m_1/n_2 \rceil \leq \lceil m_2/n_1 \rceil$, if $\min\{q_1, q_2\} < (s_1 + s_2) \cdot (\lceil m_1/n_2 \rceil - 1)$, then having two batches achieves minimizes the makespan. Otherwise, the optimal schedule is to process jobs from Families 1 and 2 alternately. In the second case, every batch of Family 1 and Family 2 except the last batch has no more than n_2 and n_1 jobs, respectively. Furthermore, Families 1 and 2 have the same number of batches of $\lceil m_1/n_2 \rceil$ jobs, and Family 1 is processed first.*

Proof. As in Lemma 5.1 only one of the two solutions in Figure 4 may be optimal. The total setup and qual-run time is $\min\{q_1, q_2\} + s_1 + s_2$ in solution (1), and $(s_1 + s_2) \cdot (\lceil m_1/n_2 \rceil - 1)$ in solution (2). Otherwise, the proof is analogous to the proof of Lemma 5.1. \square

5.2 N -family Case

This subsection considers the N -family problem with different setup and qual-run times. The goal is to obtain structural results characterizing the optimal solutions. First, new notation is introduced to facilitate the analysis:

$Q(i_1, \dots, i_k)$ represents a block of the schedule which contains jobs only from Families i_1 to i_k and for which there is no qual-run in the block.

$Q(\cdot)$ represents a block of the schedule which contains an unspecified subset of families (a “batch”) without any qual-runs. $|Q(\cdot)|$ denotes the number of jobs in $Q(\cdot)$.

$Q(\cdot, i_1, \dots, i_n)$ represents the schedule which contains some unspecified families in addition to Family i_1 to i_n , where $\{i_1, \dots, i_n\} \subset \{1, \dots, N\}$.

When there are two or more $Q(\cdot)$'s listed together, there is one qual-run and one setup between any two consecutive $Q(\cdot)$'s, but there is no qual-run for the first $Q(\cdot)$ in any schedule. For example, $Q(1, 2)Q(3)$ represents a schedule in which all the jobs from families 1 and 2 are processed in some sequence (with necessary setups between switch-overs but without any qual-run). Family 3 is processed only after all jobs of Family 1 and 2 are finished, and a qual-run for Family 3 is required before processing any job of Family 3.

Lemma 5.3. $Q(\cdot, k)Q(j)$ has a makespan no greater than that of $Q(\cdot)Q(j, k)$ if a qual-run for Family j is required in both schedules.

Proof. Since Family k is contained in the second block of $Q(\cdot)Q(j, k)$, which has only one qual-run for Family j , this indicates that the APC-value of Family k is not exceeded. Thus, there is no need to perform a qual-run for Family k . Therefore, if all the jobs of Family k in $Q(j, k)$ are appended to $Q(\cdot)$, then $Q(\cdot, k)$ has the same setup time as the original $Q(\cdot)$ plus one setup for Family k .

After appending all jobs of Family k at the end of $Q(\cdot)$, the total setups and qual-runs in $Q(\cdot, k)Q(j)$ consist of the setups for $Q(\cdot)$ plus one qual-run for Family j , and plus one setup for Family k and j , respectively. On the other hand, the setups and qual-runs in $Q(\cdot)Q(j, k)$ consist of the setups for $Q(\cdot)$ plus one qual-run for Family j , and plus the setups in $Q(j, k)$, which has at least one setup for Family k and j , respectively. Thus, the makespan of $Q(\cdot, k)Q(j)$ is not larger than that of $Q(\cdot)Q(j, k)$. \square

Lemma 5.4. $Q(\cdot, i_1, i_2, \dots, i_{k-1}, i_{k+1}, \dots, i_n)Q(i_k)$ has a makespan no greater than that of $Q(\cdot)Q(i_1, i_2, \dots, i_k, \dots, i_n)$ if a qual-run for Family i_k is required in both schedules.

Proof. Notice that in both schedules a qual-run is required for Family i_k , and no qual-run is required for Family i_j , where $j \neq k$. In terms of qual-runs and setups, $Q(\cdot)Q(i_1, i_2, \dots, i_k, \dots, i_n)$ contains one qual-run for Family i_k , the setups for $Q(\cdot)$, and the setups for $Q(i_1, i_2, \dots, i_k, \dots, i_n)$. After all the families other than i_k are appended to $Q(\cdot)$, $Q(\cdot, i_1, i_2, \dots, i_{k-1}, i_{k+1}, \dots, i_n)Q(i_k)$ contains one qual-run and one setup for Family i_k , the same setups for $Q(\cdot)$, and the setups for the appended part which contains $\{i_1, i_2, \dots, i_{k-1}, i_{k+1}, \dots, i_n\}$. Due to the fact that $\{i_1, i_2, \dots, i_{k-1}, i_{k+1}, \dots, i_n\}$ has one family less than $\{i_1, i_2, \dots, i_k, \dots, i_n\}$, the setup time of $Q(i_1, i_2, \dots, i_k, \dots, i_n)$ is no less than the setup time for the appended part plus one setup time for Family i_k .

Thus, the makespan of $Q(\cdot, i_1, i_2, \dots, i_{k-1}, i_{k+1}, \dots, i_n)Q(i_k)$ is not greater than that of $Q(\cdot)Q(i_1, i_2, \dots, i_k, \dots, i_n)$. \square

Theorem 5.5. *In the N -family setting, there exists an optimal schedule such that only the first block contains multiple families, the first block has no families in common with the later blocks, and all other blocks contain one family each, e.g., the schedule is of the form $Q(i_1, i_2, \dots, i_{N-k}) Q(i_{N-k+1}) \dots Q(i_N)$.*

Proof. For any schedule S , suppose $Q(\cdot, j)$ is the last batch which has multiple families, i.e., all batches after batch $Q(\cdot, j)$ contain one family each, and suppose the qual-run is for Family j . Since all the batches after $Q(\cdot, j)$ must have qual-runs, any changes to batch $Q(\cdot, j)$ or the batches before $Q(\cdot, j)$ do not affect the batches after $Q(\cdot, j)$. Therefore, according to Lemma 5.4, a better schedule can be obtained by moving all the jobs from families other than j out of $Q(\cdot, j)$ and into the batch right before it. A similar analysis can be applied to the last multi-family batch in the new schedule until all the batches except the first one contain one family each in the new schedule. Therefore, there exists an optimal solution where only the first block contains multiple families. \square

Theorem 5.5 greatly reduces the search space for the optimal solution, and provides the basis for the heuristic algorithm in Section 6.

6 Heuristic Algorithm

This heuristic algorithm is based on the special case analysis in Section 5. The results presented in that section do not consider machine capacity requirements. Obviously, the heuristic needs to use the insights of the previous section and consider machine capacity. Our insights indicate that a good schedule should be generated by three main steps:

1. Select the families which require a qual-run and which should be separated from the first block $Q(\cdot)$.
2. Generate the schedule for the first block $Q(\cdot)$ with the remaining families.
3. Revise the schedule to fit the machine capacity.

The major focus of the heuristic algorithm is the first block $Q(\cdot)$. The idea is to exclude families from $Q(\cdot)$ one by one according to four criteria as described in the remainder of this section, and then generate a schedule for

the first $Q(\cdot)$. According to Theorem 5.5, the families excluded from the first block require a qual-run and are appended at the end of the schedule.

Each family has four major parameters: the APC threshold value (n_i), the number of jobs in the family (m_i), the setup time (s_i), and the qual-run time (q_i). These parameters could affect other families and the whole schedule. The criteria for selecting families to be excluded from the first block is based on these four parameters.

Suppose there are N families and the problem is to decide which one of two families, Family A or Family B , should be included in the first block. Suppose $n_A < n_B$, and the other parameters such as qual-run time and number of jobs are the same for both families. Suppose Family A instead of Family B is scheduled in the first block $Q(\cdot)$. Due to the fact that there is no qual-run within a block, it is possible that other families are divided into small batches to meet the APC-value of Family A , which results in more setups than a schedule which includes Family B and excludes Family A in the first block. Therefore, when the other parameters are the same and one family has to be excluded from the first block, the family with the smaller APC-value should be separated from the first block.

Next suppose $m_A > m_B$, and all other parameters are the same for Family A and Family B . If one of them has to be excluded from the first block, it is better to choose A for exclusion. If we consider a first block with Family A jobs and one with Family B jobs instead, the latter block can be formed such that it has the same, or fewer setups than the former block. Thus, in general a family with more jobs should have higher priority to be separated from the first block than a family with fewer jobs. Similarly, when the other parameters are the same and one out of two families has to be excluded from the first block, the family with the larger setup time or the smaller qual-run time should be excluded from the first block in order to minimize the makespan. Altogether, the following heuristic guiding principles are obtained.

Heuristic principles: For the N -family setting, suppose one of two families, A or B , has to be excluded from the first block and requires a qual-run. Families A and B have the same values except for one of the following parameters: APC-value, number of jobs, setup time, or qual-run time. In order to obtain a small makespan, the family to be excluded from the first block should have one of the following characteristics (as compared to the included family): (1) smaller APC-value; (2) larger number of jobs; (3) larger setup time; or (4) smaller qual-run time.

Let F_0 denote the set of families which are included in the first block, and let $I \setminus F_0$ represent all families which are excluded from the first block. The scheme of the heuristic algorithm is described as follows. First, $F_0 = I$ and

all families are ranked according to criterion (1) of the heuristic principles. Some steps are used to generate a feasible schedule without any qual-run if it is possible to do so. Then the family with the smallest APC-value is removed from F_0 . A new first block is generated, if possible, for all families in F_0 such that a qual-run is not needed. Then, the family with the second smallest APC-value is removed from F_0 , and so on until F_0 contains only two families, which are scheduled according to Lemma 5.2. The excluded families are scheduled after the first block according to the shortest processing time criterion in order to maximize production within the machine capacity. After all these are finished, the whole process is repeated according to criterion (2), (3) and (4) in the heuristic principle, respectively. Thus, four heuristic schedules are generated. The final schedule is the one with the smallest deviation from the processing targets, where the deviation is difference between the total processing target and total number of jobs produced. If two schedules have the same deviation from the processing targets, then the one with a smaller total setup time and qual-run time is selected as the output. The detailed heuristic algorithm is as follows.

Heuristic Algorithm

All families are ranked according to the following four criteria separately:

- R-1 increasing order of number of jobs m_i
- R-2 increasing order of setup times s_i
- R-3 decreasing order of APC-value n_i
- R-4 decreasing order of qual-run time q_i .

Thus, we have four different orderings for all families, and denote them as R_l with $l = 1, \dots, 4$. For example, the family with the largest u_i has the lowest rank in R_1 , and the family with the largest n_i has the highest rank in R_3 . Starting with $l = 0$ and $F_0 = I$, the following steps are applied to each R_l .

S-1 *Initialization*: Following the structural results in Theorem 5.5, all families in F_0 are used to generate a temporary schedule for the first block, and all families in $I \setminus \{F_0\}$ are sequenced in arbitrary order after the first block, and none of them are partitioned. Step S-1(a) and S-1(b) are implemented to generate the first block $Q(\cdot)$.

- (a) If F_0 contains only two families, then generate the schedule for F_0 based on Lemma 5.2, set F_0 to the empty set, and go to S-2.

Otherwise, set $\tilde{n} = \min\{n_i, \forall i \in F_0\}$ and divide each family in F_0 into batches of size \tilde{n} .

- (b) Schedule these batches one family after another sequentially according to the ranks in R_l with the highest rank first. For example, if $F_0 = \{i, j\}$ and Family i has a higher rank than Family j , then the schedule is to process \tilde{n} jobs of Family i followed by \tilde{n} jobs of Family j , then process \tilde{n} jobs of Family i again, and so on. The pattern continues until all jobs in F_0 are scheduled. If all jobs from any Family k are exhausted during the process, update $\tilde{n} = \min\{n_i, \forall i \in F_0 \setminus \{k\}\}$. After all jobs are scheduled, go to S-2.

S-2 *Back search*: Suppose the total number of batches in the first block is B after the completion of step S-1. Let b represent batch number and i_b represent the family to which batch b belongs.

- (a) Set $b := B$, i.e., b is the last batch in the first block.
- (b) If $b = 0$, go to S-3. Otherwise, starting from the first batch of Family i_b , check if all jobs in the entire batch b can be reallocated into one or several batches of Family i_b without causing any qual-run for any other family. If it is feasible to do so, partition batch b if necessary and reallocate the jobs in batch b into those batches of Family i_b . Otherwise, set $b := b - 1$ and repeat step S-2(b).

S-3 *Cutoff*: Suppose batch b^* is the first batch that exceeds the machine capacity.

- (a) If b^* is not in the first block, then go to S-3(c). Otherwise, find the maximum number of jobs that can remain in batch b^* without exceeding the machine capacity, and delete other jobs from the original batch b^* . Calculate the remaining machine capacity, which is denoted by C_{rem} .
- (b) If the remaining capacity C_{rem} is zero, then stop and go to step S-4. If b^* is the last batch of the first $Q(\cdot)$, then go to step S-3(c). If the remaining capacity C_{rem} is greater than one setup time plus processing time of one job in batch $(b^* + 1)$, set $b^* := b^* + 1$, and go to Step S-3(a). Otherwise, set $b^* := b^* + 2$ and go to step S-3(a).
- (c) Find the maximum number of jobs to fill the the remaining capacity C_{rem} from families which are excluded from the first block.

S-4 Record the best schedule according to (1) minimum deviation from the processing targets, then (2) minimum total setup and qual-run time.

- (a) If F_0 is not empty, remove from F_0 the family which has the lowest rank of $R-l$ in F_0 , update F_0 and go to S-1.
- (b) If $r = 4$ and F_0 is empty, then output the best schedule. Otherwise set $r := r + 1$, $F_0 := I$, and go to S-1.

While it takes a very long time to solve even a small-scale IP model in Section 3, the heuristic algorithm runs very fast. It takes around 1.2 seconds for the heuristic algorithm on a Pentium M 1.6GHz laptop with a 2GB memory to find a solution for a problem with twenty families. Solving the IP model for the same problem takes several hours. The next section presents numerical results to illustrate the quality of the heuristic algorithm.

7 Numerical Study

The numerical study is performed mainly based on three-family problems to compare the heuristic method in Section 6 and the IP model in Section 3. However, the heuristic was also run on larger problems, with up to 20 families. With a larger number of families it was impossible to obtain a solution from the IP due to very long run times. All the numerical studies are performed on a laptop with a 2GB memory and a Pentium M 1.6GHz CPU. Each family parameter for each problem instance was randomly generated using the following distributions:

- m_i number of jobs in Family $i \in N$, $U[10,30]$
- s_i setup time, $U[1,21]$ minutes
- n_i APC-value, $U[3,7]$
- p_i processing time, $U[30,40]$ minutes
- q_i qual-run time, set equal to p_i , $\forall i \in N$,

where $U[a,b]$ represents a discrete uniform random variable. In semiconductor manufacturing, the qual-run time usually is the same as the wafer processing time, thus q_i is set equal to p_i in the experiments. Ten problems with randomly generated parameters are examined under each of three machine capacity levels: low, medium, and high. With the low machine capacity (1000 minutes), some processing targets can not be fulfilled. With

large machine capacity (3000 minutes), processing targets are achievable for all settings. For each problem, the IP model is run up to 4 hours using Xpress [1] and the best or the optimal integer solution (if available) is reported. The IP model finds the optimal solutions for 28 out of 30 instances. For the two instances in which the optimal solutions are not found within 4 hours, the IP model runs until the optimality gaps are less than 3% for the first phase and 10% for the second phase. Then, the best integer solutions are used as benchmarks. The heuristic algorithm finishes in less than 1 second for all the problem settings.

Tables 1-3 summarize the results from all experiments. If the heuristic algorithm achieves the same result as the IP model does, then the corresponding instance number is in **bold** font. If the IP model does not confirm the optimality of the solution, then there is an asterisk (*) right after the corresponding IP result. The makespan of the heuristic result is also listed in the tables as a reference of how the schedule fits into the limited machine capacity. The last column in the tables is the CPU time for the heuristic algorithm in units of milliseconds. The CPU time for the IP model is omitted because it takes at least one hour to find the optimal solution for most of the instances.

Table 1: Comparison between the IP model and the heuristic algorithm (machine capacity is 1000 minutes)

| # | Deviation from Target (jobs) | | Setup & Qual-run (minutes) | | Makespan (minutes) | Solution Time (milliseconds) |
|-----------|------------------------------|-----------|----------------------------|-----------|--------------------|------------------------------|
| | IP | Heuristic | IP | Heuristic | Heuristic | Heuristic |
| 1 | 26 | 26 | 22 | 32 | 996 | 0 |
| 2 | 32 | 32 | 7 | 7 | 971 | 15 |
| 3 | 35 | 35 | 48 | 50 | 995 | 31 |
| 4 | 39 | 40 | 36 | 68 | 999 | 19 |
| 5 | 52 | 52 | 24 | 24 | 986 | 15 |
| 6 | 39 | 39 | 33 | 33 | 968 | 31 |
| 7 | 23 | 23 | 54 | 54 | 978 | 31 |
| 8 | 30 | 30 | 49 | 49 | 978 | 16 |
| 9 | 12 | 12 | 60 | 60 | 958 | 15 |
| 10 | 36 | 36 | 58 | 58 | 975 | 32 |

Table 2: Comparison between the IP model and the heuristic algorithm (machine capacity is 2000 minutes)

| # | Deviation from Target (jobs) | | Setup & Qual-run (minutes) | | Makespan (minutes) | Solution Time (milliseconds) |
|-----------|------------------------------|-----------|----------------------------|-----------|--------------------|------------------------------|
| | IP | Heuristic | IP | Heuristic | Heuristic | Heuristic |
| 1 | 1 | 1 | 82 | 82 | 1986 | 0 |
| 2 | 7* | 7 | 87* | 94 | 1984 | 47 |
| 3 | 10 | 10 | 91* | 91 | 1982 | 31 |
| 4 | 13 | 13 | 117 | 117 | 1981 | 47 |
| 5 | 24 | 24 | 71 | 72 | 1976 | 47 |
| 6 | 15 | 15 | 121 | 121 | 1986 | 15 |
| 7 | 0 | 0 | 101 | 101 | 1807 | 31 |
| 8 | 6 | 6 | 97 | 97 | 1983 | 46 |
| 9 | 0 | 0 | 94 | 94 | 1412 | 15 |
| 10 | 10 | 10 | 80 | 80 | 1999 | 16 |

Table 3: Comparison between the IP model and the heuristic algorithm (machine capacity is 3000 minutes)

| # | Deviation from Target (jobs) | | Setup & Qual-run (minutes) | | Makespan (minutes) | Solution Time (milliseconds) |
|-----------|------------------------------|-----------|----------------------------|-----------|--------------------|------------------------------|
| | IP | Heuristic | IP | Heuristic | Heuristic | Heuristic |
| 1 | 0 | 0 | 82 | 82 | 2026 | 16 |
| 2 | 0 | 0 | 94 | 94 | 2250 | 47 |
| 3 | 0 | 0 | 100 | 100 | 2391 | 16 |
| 4 | 0 | 0 | 117 | 117 | 2501 | 16 |
| 5 | 0 | 0 | 104 | 104 | 2944 | 31 |
| 6 | 0 | 0 | 121 | 121 | 2577 | 16 |
| 7 | 0 | 0 | 101 | 101 | 1807 | 16 |
| 8 | 0 | 0 | 107 | 107 | 2233 | 16 |
| 9 | 0 | 0 | 94 | 94 | 1412 | 0 |
| 10 | 0 | 0 | 102 | 102 | 2421 | 15 |

When the machine capacity is 1000 minutes, the heuristic algorithm finds the optimal solutions in 7 out of 10 instances. When the machine capacity is 2000 minutes, the heuristic algorithm achieves the same results as the IP model does in 8 instances, and 7 of them are proved to be optimal. When the machine capacity is 3000 minutes, all jobs can be done within the machine capacity for all 10 instances. The heuristic algorithm finds the optimal

solutions for all 10 settings in at most 47 milliseconds CPU time. Based on these observations, it can be concluded that the heuristic algorithm works quite well.

Table 4: Percentage across all the experiments to find the best solution for different ranking rules in the heuristic algorithm

| | APC-value n_i | Qual-run q_i | Number of Jobs m_i | Setup s_i |
|------------|-----------------|----------------|----------------------|-------------|
| Percentage | 33% | 90% | 67% | 17% |

As described in Section 6, four different ranking rules are used to select the family to be excluded from the first block in the heuristic algorithm. They are restated here:

1. decreasing order of the APC-value (n_i)
2. decreasing order of the qual-run time (q_i)
3. increasing order of the number of jobs (n_i)
4. increasing order of the setup time (s_i).

In any given experiment, different rules may result in the same best (possibly suboptimal) schedule. In order to see which of these rules works best, the frequency of finding the best schedule for each rule is also recorded. Table 4 gives the percentage of time each rule gave the best solution. Due to the limited number of experiments, such a summary only serves as a general indication of the relative performance of the ranking rules. Table 4 indicates that there is a good chance of finding the best solution by selecting the families according to qual-run time (in decreasing order) or the number of jobs (in increasing order). Such an observation could be helpful in creating dispatching rules for APC scheduling in the future.

Finally, Figure 5 shows the CPU time (in milliseconds) used by the heuristic algorithm to solve problems with up to 20 different product families. The parameter settings were again randomly generated as described at the beginning of this section. Each point in Figure 5 represents one instance. As the number of families increases, it seems that the CPU time increases linearly, and it takes only around 1.2 seconds to find the solution even with 20 different families. Thus, such a heuristic algorithm could be very useful in practice.

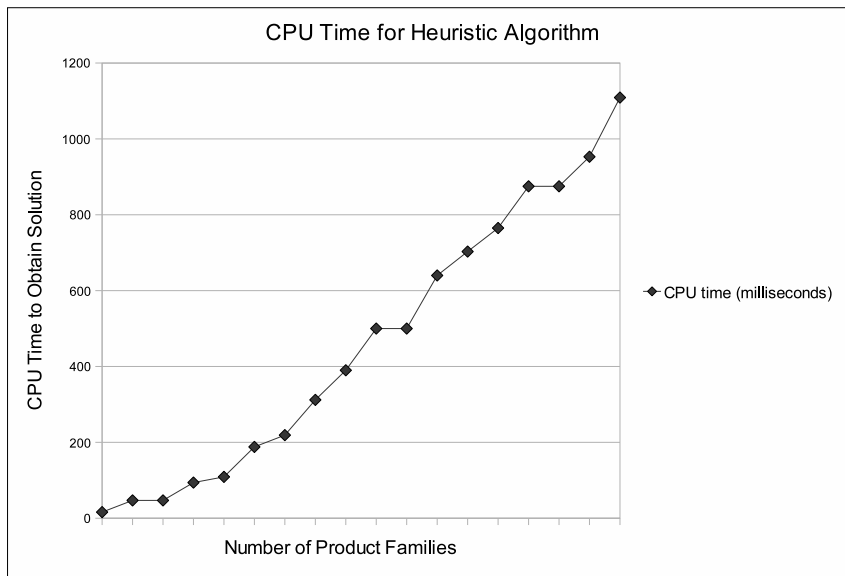


Figure 5: CPU time used by the heuristic algorithm

8 Conclusion

This paper explores a new problem inspired by issues in semiconductor manufacturing: scheduling with APC constraints. This paper studies the complexity of two versions of the constraints: count-based and time-based APC constraints, and both versions are proved to be strongly NP-hard.

As an initial step, this paper only studies the single-machine version of the problem. Optimal solutions are found for the two-family case, and the structure of the optimal solutions is analyzed for the N -family case. These properties are utilized in the heuristic algorithm to find a good solution. In order to evaluate the heuristic algorithm, we study an IP formulation and a goal-programming approach is implemented to search for the optimal solution. The comparison between the heuristic algorithm and the integer program shows that the proposed heuristic algorithm can find very good solutions extremely quickly. Additional numerical experiments indicate that even with twenty different families, the heuristic algorithm takes around 1 second to find a solution.

Future research work would consider the APC scheduling problem in a parallel machine setting. In a parallel-machine model, the allocation of different families to different machines could be a key factor to effectively avoid qual-runs. Also, another attractive topic is to consider incoming jobs when making APC-scheduling decisions.

A Appendix

A.1 Proof of Lemma 4.1

Proof. First of all, there are unlimited choices to satisfy the conditions imposed by the lemma. For example, $p_i = b/3, \forall i \in S$. Arbitrary positive values can be assigned to s_i ($\forall i \in \{0, \dots, 3t\}$). Then, T_0 can be determined by condition (13). Choose p_0 such that $p_0 > 2T_0$, and then choose arbitrary values between $(2T_0 + p_0, 2p_0)$ for T_i ($\forall i \in S$). Finally, condition (15) can be used to calculate q_i ($\forall i \in \{0, \dots, 3t\}$).

In the following, it is first proved that the makespan of any schedule with qual-runs is greater than (16). Then, it is proved that in order to avoid any qual-run, conditions (T-1) and (T-2) must hold.

We now prove the assertion regarding the minimum makespan. In this case, (16) can be achieved if all other families do not need any qual-run and only one setup for each family is necessary. Then, the makespan is:

$$\begin{aligned}
 & \sum_{i=0}^{3t} s_i + (t+1) \sum_{i=0}^{3t} p_i + q_i \\
 &= \left\{ t \sum_{i=0}^{3t} s_i + (t+1) \sum_{i=0}^{3t} p_i \right\} + q_i - (t-1) \sum_{i=0}^{3t} s_i \\
 &> t \cdot \sum_{i=0}^{3t} s_i + (t+1) \sum_{i=0}^{3t} p_i.
 \end{aligned}$$

The last inequality is due to constraint (15). Thus, if there is any qual-run in the schedule, the makespan is greater than (16).

In order to avoid Family 0 qual-run, the first job of Family 0 must start at α_0 , where $0 \leq \alpha_0 \leq T_0$ and the interval between the processing of two consecutive jobs of Family 0 has to be less than or equal to T_0 . On the other hand, if two jobs of Family 0 are processed consecutively, then any other family processed after these two jobs will require a qual-run, because of condition (14). Therefore, no two jobs of Family 0 can be processed consecutively before the jobs of all other families are finished. If no job of Family i ($i \neq 0$) is processed between two jobs of Family 0 before $\alpha_0 + t(p_0 + T_0)$, then the interval between any two jobs of Family i would be greater than $2p_0$, inducing a qual-run due to condition (14). Thus, at least one job of Family i ($\forall i \in S$) needs to be processed in the time interval of $[\alpha_0 + p_0 + k(p_0 + T_0), \alpha_0 + p_0 + k(p_0 + T_0) + T_0]$, where $k = 0, \dots, t-1$.

If the jobs of Family 0 are scheduled according to condition (T-1) and thus $\alpha_0 = T_0$, then there are t intervals as defined in condition (T-2). The

jobs are reallocated at point B , then they need qual-runs due to the previous analysis. Therefore, wherever the jobs in \tilde{S} are reallocated, the resulting schedule has a makespan which is greater than (16). Thus, all intervals must have a length of T_0 , and we must have $\alpha_0 = T_0$ in order to achieve makespan (16).

So far, it has been proved that in order to achieve a makespan no greater than (16), the schedule must satisfy the following conditions: (1) the first job of Family 0 starts at time T_0 , (2) the interval between every two jobs except the last two jobs of Family 0 is T_0 , and (3) at least one job of Family i ($\forall i \in S$) needs to be processed within the time interval of $[k(p_0 + T_0), (k + 1)(p_0 + T_0) + T_0]$, where $k = 1, \dots, t - 1$. In order to meet all three conditions, the schedule must be in the form of Figure 2. After all jobs of Family 0 are scheduled, and t jobs of Family i ($i \in S$) are scheduled, the remaining available time is

$$T_0 - \sum_{i=1}^{3t} (s_i + p_i) = T_0 - t \cdot b - \sum_{i=1}^{3t} s_i = b,$$

in each interval $[k(p_0 + T_0), (k + 1)(p_0 + T_0) + T_0]$, where $k = 1, \dots, t - 1$. One job of Family i ($\forall i \in S$) needs to be allocated into the remaining slots in each interval. Since $\{S_j | j = 1, \dots, t\}$ is a partition of $\{p_1, \dots, p_{3t}\}$ that solves the 3-partition problem such that $\sum_{i \in S_j} p_i = b$, then the jobs in S_k can

be scheduled in interval k .

Note that the families in the interval $[k(p_0 + T_0), k(p_0 + T_0) + T_0]$ can be arranged in any sequence without causing qual-runs for other families. Also, the makespan is not changed as long as there is only one setup for each family within that interval. A job of Family i_{kj} ($i_{kj} \in S_k$) can be scheduled to immediately follow the already scheduled job of Family i_{kj} in interval k , and no additional setup is required. Therefore, the final schedule A_t^* must satisfy conditions (T-1) and (T-2).

Since there is no qual-run in A_t^* , the only way to reduce the makespan is to reduce setup time. However, there have to be at least t setups for each family due to (T-1) and (T-2). Therefore, the minimum makespan is the sum of all processing times and necessary setup times, which is:

$$\begin{aligned} & \sum_{i=0}^{3t} (m_i - 1) s_i + \sum_{i=0}^{3t} m_i p_i \\ &= t \sum_{i=0}^{3t} s_i + (t + 1) \sum_{i=0}^{3t} p_i. \end{aligned}$$

Thus the minimum makespan is (16). □

A.2 Proof of Lemma 4.3

Proof. First of all, if any Family i has a qual-run, then the minimum makespan is achieved when there is no qual-run for any family except Family i , and every family has only one setup. The corresponding makespan is:

$$z_i = \sum_{j=0}^{3t} s_j + \sum_{j=0}^{3t} m_j \cdot p_j + q_i = z - (t-1) \cdot s_0 + q_i.$$

However, $z - (t-1) \cdot s_0 + q_i$ is greater than z due to condition (20). Thus, if any family has a qual-run, then the makespan z is not achievable. Below, it is proved that any schedule must satisfy (C-1) and (C-2) in order to avoid having any qual-runs.

Any schedule A must have one of the following two mutually exclusive properties:

- (i) There exists at least one partition for at least one Family i for $i \in S$;
or
- (ii) There is no partition for any family in $S = \{1, 2, \dots, 3t\}$.

In case (i), suppose Family r ($r \in S$) is partitioned into at least two parts. Then, at least one additional setup of Family r is required with setup time s_r . The total processing time for all jobs from all families is at least

$$z' = \sum_{i=0}^{3t} s_i + \sum_{i=0}^{3t} m_i \cdot p_i + s_r > z.$$

The inequality is due to condition (21). Therefore, the makespan of any schedule in case (i) is greater than z .

In case (ii), any Family i ($i \in S$) should start processing before position n_i to avoid any qual-run. Since there is no partition for Family i , the last job of Family i would be processed before or at position $(n_i + m_i - 1)$ (see Figure 7). Notice that:

$$n_i + m_i - 1 = (tn_0 - m_i) + m_i - 1 = t \cdot n_0 - 1.$$

Since $\sum_{i \in S} m_i = t(n_0 - 1)$, it is possible to schedule all jobs from the families in S and $(t-1)$ jobs from Family 0 before position $t \cdot n_0$.

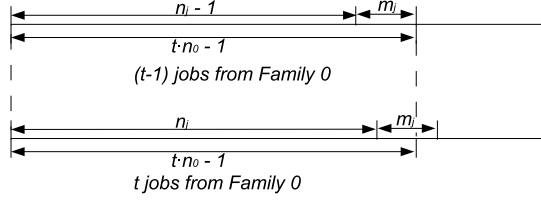


Figure 7: No more than $t - 1$ jobs from family 0 before position $(t \cdot n_0 - 1)$

Due to the fact that all jobs from the families in S must be scheduled before position $t \cdot n_0$, no more than $(t - 1)$ jobs of Family 0 can be scheduled before position $t \cdot n_0$. Therefore, the remaining jobs of Family 0 need to be scheduled after position $t \cdot n_0$. In order to avoid qual-run for the job of Family 0 at position $t \cdot n_0$, the last job of Family 0 before position $t \cdot n_0$ must be scheduled somewhere between $(t \cdot n_0 - n_0)$ and $(t \cdot n_0 - 1)$.

On the other hand, in order to avoid qual-run, the first job of Family 0 must be scheduled at position α , where $\alpha \leq n_0$. If there are $(t - 1)$ jobs of Family 0 before position $t \cdot n_0$, and they are scheduled every n_0 positions to avoid qual-run, then the last job of Family 0 before position $t \cdot n_0$ is $[\alpha + (t - 2)n_0]$. However, the last job of Family 0 before position $t \cdot n_0$ must be scheduled between $(t \cdot n_0 - n_0)$ and $(t \cdot n_0 - 1)$. Thus,

$$\begin{aligned} t \cdot n_0 - n_0 &\leq \alpha + (t - 2)n_0 \\ \Rightarrow \alpha &\geq n_0. \end{aligned}$$

Since $\alpha \leq n_0$ and $\alpha \geq n_0$, α must equal n_0 . Also, it is proved that in order to avoid a qual-run for Family 0, at least $(t - 1)$ jobs of Family 0 must be scheduled before position $t \cdot n_0$. Otherwise, there is no way to schedule jobs of Family 0 at least every n_0 positions to avoid qual-run. Therefore, exactly $(t - 1)$ jobs of Family 0 must be scheduled before position $t \cdot n_0$, and the schedule satisfies (C-1). Also, since $\{S_j \mid j = 1, \dots, t\}$ is a partition of $\{m_1, \dots, m_{3t}\}$ such that $\sum_{i \in S_j} m_i = n_0 - 1$, all jobs of the families in S can

be allocated into those intervals between Family 0 jobs before position $t \cdot n_0$. Thus, the schedule satisfies (C-2).

Altogether, it has been shown that the optimal schedule must satisfy (C-1) and (C-2), and the corresponding makespan is greater than or equal to (22). \square

Acknowledgements This research was supported by a National Science Foundation grant under DMI-0432433 and the members of the Texas-Wisconsin-

California Control Consortium.

References

- [1] *Xpress-Mosel User Guide*. Dash Optimization Ltd., Englewood Cliffs, New York, 2004.
- [2] R. P. Good and S. J. Qin. On the stability of mimo ewma run-to-run controllers with metrology delay. *IEEE Transactions On Semiconductor Manufacturing*, 9(1):78–86, 2006.
- [3] C. A. Harrison, R. P. Good, Kadosh D., and S. J. Qin. A multi-step supervisory control strategy for semiconductor device manufacturing. *43rd IEEE Conference on Decision and Control*, 2004.
- [4] C. L. Monma and C. N. Potts. On the complexity of scheduling with batch setup times. *Operations Research*, 37(5):798–804, 1989.
- [5] A. J. Pasadyn and T. F. Edgar. Observability and state estimation for multiple product control in semiconductor manufacturing. *IEEE Transactions On Semiconductor Manufacturing*, 18(4):392–604, 2005.
- [6] S. J. Qin, Gregory Cherry, R. P. Good, J. Wang, and C. A. Harrison. Semiconductor manufacturing process control and monitoring: A fab-wide framework. *Journal of Process Control*, 16:179–191, 2006.
- [7] S. J. Qin, G. W. Scheid, and T. J Riley. Adaptive run-to-run control and monitoring for a rapid thermal processor. *Journal of Vacuum Science & Technology B*, 21(1):301–310, 2003.
- [8] L. B. Toktay and R. Uzsoy. A capacity allocation problem with integer side constraints. *European Journal of Operational Research*, 109:170–182, 1998.
- [9] R. Uzsoy and J. D. Velásquez. Heuristics for minimizing maximum lateness on a single machine with family-dependent set-up times. *Computers and Operations Research*, 35(6):2018–2033, 2008.
- [10] S. Webster and K. R. Baker. Scheduling groups of jobs on a single machine. *Operations Research*, 43(4):692–703, 1995.
- [11] W. Yang and C Liao. Survey of scheduling research involving setup times. *International Journal of Systems Science*, 30(2):143–155, 1999.

- [12] J. J. Yuan, Z. H. Liu, C. T. Ng, and T. C. E. Cheng. Single machine batch scheduling problem with family setup times and release dates to minimize makespan. *Journal of Scheduling*, 9:499–513, 2006.
- [13] S. Zdrzalka. A sequencing problem with family setup times. *Discrete Applied Mathematics*, 66:161–183, 1996.