An efficient approach to cross-fab route planning for wafer manufacturing

Muh-Cherng Wu*, Chang-Fu Shih, Chen-Fu Chen

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

1. Introduction

Recently, some semiconductor companies tend to adopt a dual-fab strategy. The strategy advocates that building two fabs at a time, which are next door to each other. One reason why this strategy arises is due to that semiconductor equipment, compared with fab space, is much more expensive and shorter in lead time of acquisition. In practice, more than 80% expenditure of an advanced wafer fab is spent on equipment. The acquisition lead time for equipment ranges from 3 to 9 months, while that for fab space takes about 1–2 years. To quickly respond to volatile market demand, some companies tend to build a large scale space for two fabs in advance and gradually purchase equipment based on future demand over time.

Not only good in fast capacity expansion, the dual-fab strategy also provides a capacity-sharing mechanism due to close proximity of the two fabs. Consider a single-fab production policy which requests each wafer job be manufactured in only one fab. In a dual-fab configuration, such a policy usually leads to underutilization of equipment because idle equipment capacity in one fab cannot be used by the other. To utilize the idle capacity, we may have to adopt a cross-fab production policy. That is, the manufacturing of a wafer job could be partly done in one fab and partly in the other.

However, under the cross-fab production policy, we would be confronted with a route planning problem—how to appropriately assign the operations of a wafer job to each of the two fabs. Prior studies on such a route planning problem are relatively few. With each job route being cut into several segments, Toba, Izumi, Hatada, and Chikushina (2005) studied the route planning problem in a real-time manner. That is, whenever a segment is completed, a decision—which fab to manufacture the next segment—must be immediately made. Wu and Chang (2007) examined a route planning problem in a weekly horizon. Assuming the two fabs plan capacity exchange weekly, they attempted to find an optimal capacity-trading portfolio in order to maximize the total throughput of the two fabs.

Aside from the track of short-term route planning, Wu, Erkoc, and Karabuk (2005) addressed the problem from a relatively long-term perspective. Given a product mix to produce, say in a quarter, they attempted to determine how to cut the route of each product into two segments; and determine the production ratio of each segment that should be assigned to each fab. Their objective function is to maximize the total throughput of the two fabs subject that a target cycle time must be met.

Numeric experiments indicated that the method proposed by Wu, Chen, and Shih (2008) could effectively increase equipment utilization and total throughput for a dual-fab scenario. However, their method may become computationally extensive in dealing with large scale cases.

In order to efficiently solve the route planning problem, this paper presents an enhanced approach based on Wu et al. (2008). Numeric experiments indicated that solutions obtained by the enhanced approach are almost as good as that obtained by Wu et al. (2008) yet requires much less computational efforts.

The remainder of this paper is organized as follows. Section 2 reviews relevant literature. Section 3 explains the route planning problem. Section 4 outlines the LP–GA solution framework pro-
posed by Wu et al. (2008). Section 5 presents the linear program (LP) and our enhancements to reduce computational time. Section 6 presents the genetic algorithm (GA) and our enhancements to reduce computational time. Numerical experiments are in Section 7 and concluding remarks are in the last section.

2. Relevant literature

In a company with multiple manufacturing sites, planners would face a capacity allocation decision—how to allocate a given demand to each manufacturing site. Literature on the capacity allocation problem could be grouped in two categories: product-level and operation-level.

In the product-level category, most literature assumed a single-site production policy—each product should be completely manufactured with a single-site. A literature survey has been published by Wu et al. (2005), and some recent studies can be referred to Chiou, Guo, Chen, Cheng, and Chen (2007), Lee, Chung, Lee, and Kang (2006) and ManMohan (2005). Most of these prior studies used the linear programming (LP) technique to solve the capacity allocation problems.

In the operation-level category, most literature assumed a cross-site production policy manufacturing operations for a product could be distributed among different sites. The need of studying the cross-site-route planning problems thus arises. Such route planning problems were mostly addressed in the context of group technology (GT). Example literature includes Dimopoulou (2006), Kim, Beak, and Jun (2005), Madhavi, Rezaeian, Shanker, and Amir (2006), Nskanda, Diaby, and Price (2006), Spiliopoulos and Sofinopoulos (2007), Vin, Lit, and Delchambre (2005).

In GT, each site is a manufacturing cell and multiple cells from a factory. A GT cell is designed for manufacturing a particular group of products, and by nature is functionally limited. A cell thus may need to outsourcing decision of each cell.

By contrast, in the route planning problem we address, each of the two fabs is assumed to be functionally comprehensive. Each product can be completely manufactured in either one of the two fabs. The purpose of cross-fab route planning is to maximize the aggregate throughput through optimum capacity-sharing.

3. Problem statement

The dual-fab route planning problem is explained in more detail, where the two fabs are called Fab_1 and Fab_2. We first present the assumptions, and proceed to the decision variables, objective function and constraints.

Assumption 1: Each fab is functionally comprehensive. Both fabs are so comprehensively equipped that each fab can individually complete the production of each product—not requiring support of the other fab.

Assumption 2: The transportation path between any two workstations/buffer is unique, rather than multiple. In practice, there exist multiple paths in transporting a wafer job from a workstation/buffer to another. However, to reduce the problem complexity, we assume that a fixed path is predefined for such a transport.

Assumption 3: Each product has only four possible routes. The process route of each product is cut into two segments, where a route's break point is called a cut-off point. A product has four possible manufacturing routes: 1 → 2, 2 → 2, 1 → 2, and 2 → 1, where notation i → j denotes the first segment is manufactured at Fab_i while the second is manufactured at Fab_j.

Define \( \pi_i = [a, b, c, d] \) as the percentage of the four possible routes of product i. Each element of \( \pi_i \) in sequence represents the route percentage of 1 → 1, 2 → 2, 1 → 2, and 2 → 1. Define \( \Pi_i \) as the cut-off point for the route of product i, which is the identification code (an integer) of the operation for separating a route into two segments. The range of \( \Pi_i \) is 1 ≤ \( \Pi_i \) ≤ \( o_i \) − 1 where \( o_i \) is the total number of operation of product i. We set \( \Pi_i \) = 0 while we determine not to manufacture product i by using any cross-fab routes.

Consider a dual-fab company that has n products to manufacture, represent a solution of the route planning problem by \( (\Pi_i, R_i) \), where \( \Pi_i = [\pi_1, \ldots, \pi_n] \) and \( R_i = [r_1, \ldots, r_n] \). The objective is to find an optimal solution \( (\Pi_i, R_i) \) in order to maximize the total throughput of the two fabs, subject to the constraint of meeting a target cycle time.

4. Solution framework

To solve the dual-fab route planning problem, we adopted the solution framework proposed by Wu et al. (2008), and developed several enhancements to their solution method in order to reduce computational efforts. As shown in Fig. 1, the solution framework involves two modules.

In Module 1, each path is assumed to be equipped with infinite transportation capacity; and the transportation time between any two workstations/buffers is thus zero. The problems so simplified are solved by an iterative use of a linear program (LP) model. This module is intended to find an optimum solution \( (\Pi_i, R_i) \), in terms of minimizing the total number of inter-fab transportation. In the prior study (Wu et al., 2008), this module is very computationally extensive because the number of LP iterations is quite huge for large scale cases. We proposed three heuristic methods to enhance the prior study by significantly reducing the number of LP iterations.

Let \( (\Pi_i, R_i) \) represent the solution obtained in Module 1. In Module 2—by taking \( \Pi_i \) as given parameters, we deal only with decision variable \( R \) by considering each path as a tool with limited transportation capacity. The transportation time for a path depends upon the traffic flow intensity. The higher the traffic intensity, the longer is the cycle time. The performance of a particular \( (\Pi_i, R) \) could be evaluated by applying a queueing network model (Connors, Feigin, & Yao, 1996). In the prior study (Wu et al., 2008), they developed a GA to find a near-optimal solution from the space \( \{(\Pi_i, R)\} \), which is also computationally extensive while dealing with large scale cases. We enhanced the prior study by reducing the size of the GA chromosomes. In the enhanced GA, many elements in \( R \) are considered to be constant and only a few need to be searched.

The essence of these two modules are compared below. Module 1 essentially deals with a static capacity allocation problem which does not consider job flow time. In contrast, Module 2 deals with a time-phased capacity allocation problem, in which job flow time is addressed and computed by a queueing network model.

![Fig. 1. Solution framework.](image-url)
5. Module 1 – LP model and enhancements

Obtaining the solution for Module 1 is through an iterative use of an LP model. We first describe the LP model, and then present the architecture of the iterative process. Finally, we describe the three methods designed to reduce the number of LP iterations.

5.1. LP model

**Indices**
- \( i \) \text{ index of product}
- \( g \) \text{ index of workstation in Fab}_1
- \( h \) \text{ index of workstation in Fab}_2

**Parameters**
- \( n \) \text{ total number of products}
- \( \pi_i \) \text{ cut-off point for defining the cross-fab routes of product } i
- \( \Pi \) \text{ } \Pi = [\pi_i], 1 \leq i \leq n, \text{ the cut-off points of all products}
- \( V \) \text{ estimated total throughput of the two fabs, input by user}
- \( z_i \) \text{ percentage of product } i \text{ in the given product mix, } \sum_{i=1}^{n} z_i = 1, 0 < z_i < 1
- \( C_g \) \text{ available machine hours of workstation } g \text{ in Fab}_1
- \( C_h \) \text{ available machine hours of workstation } h \text{ in Fab}_2
- \( m_1 \) \text{ total number of workstations in Fab}_1
- \( m_2 \) \text{ total number of workstations in Fab}_2
- \( W_g^i \) \text{ total processing time per lot required on workstation } g \text{ in Fab}_1, \text{ while product } i \text{ is manufactured by route } 1 \text{ to } 2
- \( W_g^d \) \text{ total processing time per lot required on workstation } g \text{ in Fab}_1, \text{ while product } i \text{ is manufactured by route } 2 \text{ to } 1
- \( W_h^d \) \text{ total processing time per lot required on workstation } h \text{ in Fab}_2, \text{ while product } i \text{ is manufactured by route } 1 \text{ to } 2
- \( W_h^o \) \text{ total processing time per lot required on workstation } h \text{ in Fab}_2, \text{ while product } i \text{ is manufactured by route } 2 \text{ to } 1

**Decision variables**
- \( R \) \text{ } R = \{R_1, \ldots, R_n\}, \text{ where } R_i = [a_i, b_i, c_i, d_i]
- \( a_i \) \text{ percentage of using route } 1 \text{ to } 1 \text{ in producing product } i
- \( b_i \) \text{ percentage of using route } 2 \text{ to } 2 \text{ in producing product } i
- \( c_i \) \text{ percentage of using route } 2 \text{ to } 1 \text{ in producing product } i
- \( d_i \) \text{ percentage of using route } 1 \text{ to } 2 \text{ in producing product } i

The LP model is to compute an optimum \( R \) for a given pair of \((V, \Pi)\), in terms of minimizing the number of cross-fab transportation. Define the objective function by \( Z(V, \Pi) \). The LP model is formulated below:

\[
\begin{align*}
\text{Min } Z(V, \Pi) & = \sum_{i=1}^{n} V \cdot z_i \cdot (c_i + d_i) \\
\text{s.t. } & a_i + b_i + c_i + d_i = 1, \quad 1 \leq i \leq n, \quad (1) \\
& \sum_{i=1}^{n} V \cdot z_i \cdot (a_i \cdot W_g^i + d_i \cdot W_g^d + c_i \cdot W_g^o) \leq C_g, \quad 1 \leq g \leq m_1, \quad (2) \\
& \sum_{i=1}^{n} V \cdot z_i \cdot (b_i \cdot W_h^i + d_i \cdot W_h^d + c_i \cdot W_h^o) \leq C_h, \quad 1 \leq h \leq m_2. \quad (3)
\end{align*}
\]

The objective function is to minimize the number of cross-fab production lots. The rationale for defining this objective is that cross-fab production requires longer transportation time than within-fab production. Subject to a target cycle time, an attempt to minimize cross-fab production lots tends to increase total throughput. Constraint (1) describes the dependent relationship among the route ratios. Constraints (2) and (3) ensure that the capacity used in each workstation, in \textit{Fab}_1 and \textit{Fab}_2, should be lower than its available supply. Notice that \( V \) is the estimated throughput; the LP may yield no solution while \( V \) is too large.

In the above LP model, each of the \( n \) products is eligible for cross-fab production. To reduce computational complexity, we propose to divide the products into two sets: \( Q_s \) and \( Q_c \). Products in \( Q_s \) are eligible for cross-fab production, and those in \( Q_c \) are only allowed for single-fab production. To deal with such a general scenario \((Q_s, Q_c)\), the above LP should be modified by including the following constraints:

\[
c_k = 0 \quad \text{and} \quad d_k = 0 \quad \text{for each product } k \text{ in } Q_c, \quad (4)
\]

A procedure \( LP\_Module(V, \Pi, Q_s, Q_c) \) is defined below to facilitate explaining the iterative procedures for calling the modified LP.

**Procedure LP\_Module** \((V, \Pi, Q_s, Q_c)\)

1. **Step 1**: Compute \( LP(V, \Pi, Q_s, Q_c) \)
2. **Step 2**: If (LP has no solution) then Pass\_Check = “Fail”, Return
   If (LP has solution) then Pass\_Check = “Pass”
   \( \text{Return } Z(V, \Pi), R(V, \Pi), \text{Pass\_Check} \)

In Step 1 of the above procedure, \( LP(V, \Pi, Q_s, Q_c) \) denotes the modified LP. In Step 2, Pass\_Check is a flag in which “Fail” denotes the value of \( V \) is too large. Moreover, \( Z(V, \Pi) \) denotes the obtained route ratio and \( Z(V, \Pi) \) denotes the obtained value in objective function.

5.2. Iterative process of LP

To solve the route planning problem, we need to iteratively run \( LP\_Module(V, \Pi, Q_s, Q_c) \). The architecture of the iterative process is shown in Fig. 2. The architecture involves four procedures, which are organized in a hierarchical manner. The bottom level of the hierarchy is the \( LP\_Module(V, \Pi, Q_s, Q_c) \). Details of the other three procedures are presented in Appendices 1–3.

Of the three top level procedures, \( Route\_Planning \) is intended to ask users input \((Q_s, Q_c)\) and \((L, U)\) which is the range of \( V \). Given a scenario \((L, U, Q_s, Q_c)\), \( Route\_Planning\_for\_Given\_Throughput \) is
intended to find an optimal \( V \in (L, U) \), where the algorithm for identifying \( V \) is based on a binary-search method (Fig. 3). Performance Evaluation is intended to find \( (P^*, R^*) \) for a given scenario \((V, Q_s, Q_r)\), based on a binary-search algorithm over multiple intervals and each interval is a product route.

Assume set \( Q_s \) has \( n_s \) products; that is, there are \( n_s \) product routes to search for their optimal cut-off points. The computational complexity of the iterative process is \( O(2^{n_k}) \), where \( n_k \) is a constant which denotes the maximum number of search required to carry out on each product route. It might be very much computationally extensive while \( n_k \) is high (i.e., all products are eligible for cross-fab production). One way to efficiently solve the route planning problem is to find an appropriate \( Q_s \), which has small value of \( n_k \) and can yield a good quality solution.

5.3. Reduction of iteration number

To find such an appropriate \( Q_s \), we developed a procedure Product Sorting to categorize all products into three groups. Taking each group as a particular selection of \( Q_s \), we would have three different versions of \( Q_s \). The procedure is presented below.

**Procedure Product Sorting**

Step 1: Identify the bottleneck workstation (say, \( B \)) of the two fabs.
Step 2: Compute the workload of each product on \( B \).
Step 3: Sort all the \( n \) products according to their workload on \( B \).
Step 4: Categorize products into three groups, based on the sorted results.

With three different versions of \( Q_s \), we could have three solution methods in Module 1. The method, using the product group with the highest bottleneck workload, is called \( LP_1 \). The one with middle-level bottleneck workload is called \( LP_2 \), and the remaining one is called \( LP_3 \). The method proposed by Wu et al. (2008) is called \( GA_0 \).

The rationale for taking bottleneck workload as the criterion for grouping products is two-fold. First, the utilization of bottleneck workstation dominates the two fabs’ throughput. Thus, in cross-fab route planning decisions, the capacity allocation of bottleneck workstation would be most critical. Second, we attempt to justify which product group is most critical in the cross-fab route planning—the heavily loaded group, the middle-level load group, or the lightly loaded group.

6. Module 2 – GA

Define the solution of Module 1 as \( (P^*_1, R^*_1) \), which is obtained under the assumption of infinite transportation capacity. In Module 2, with \( (P^*_1, R^*_1) \) being available, we developed a GA in order to find a better solution \( (P^*_2, R^*_2) \) under the assumption of finite transportation capacity.

The GA is an enhanced version of the one proposed by Wu et al. (2008). Like Wu et al. (2008), we first set \( P^*_2 = P^*_1 \) and attempt to find \( R^*_2 \). But in the search of \( R^*_2 \), we make an enhancement by setting \( c_t = d_t = 0 \) for each product \( k \) in \( Q_s \) (i.e., the single-fab production policy presumed in Module 1 is preserved).

The enhancement could simplify the representation of a solution. Consider a chromosome (a possible solution) represented by a vector \( R = (r_1, \ldots, r_n) \), where \( r_t = (a_t, b_t, c_t, d_t) \). We call \( r_t \) a gene-segment, and each element in \( r_t \) a gene. Since \( a_t + b_t + c_t + d_t = 1 \), we have three free genes for each product in \( Q_s \) and one free gene for each product in \( Q_r \). Here, a free gene is one whose value is changeable in the search process, while a gene whose value is not changeable is called a static gene. With this enhancement, a chromosome has only \( 3n_s + (n - n_s) \) free genes, rather than \( 3n \) ones as in Wu et al. (2008). The GA proposed by Wu et al. (2008) is called \( GA_0 \) and our enhanced version is called \( GA_1 \).

The performance (also called fitness) of each chromosome is computed by a queueing network model (Wu et al., 2008), which is adapted from the one developed by Connors et al. (1996). For a given chromosome (i.e., a route plan), the queuing network can be used to compute the aggregate throughput of the two fabs subject to meeting a target cycle time.

The GA is an iterative algorithm which can be briefly described as follows:

**Procedure GA**

Step 1: Initialization
- \( t = 0 \), Status = ‘Not-terminate’
- Randomly generate \( N_p \) chromosomes to form a population \( P_0 \)

Step 2: Genetic Evolution
While (Status = ‘Not-terminate’) do
- Use a cross-over operator to create \( N_c \) new chromosomes
- Use a mutation operator to create \( N_m \) new chromosomes
- Form a pool by taking the union of \( P_t \) and the set of newly created chromosomes
- \( t = t + 1 \), and select the best \( N_p \) chromosomes from the pool to form \( P_t \)
- Check if termination condition is met; if yes, set Status = “Terminate”

Endwhile
Step 3: Set the best chromosome in \( P_t \) as \( R^*_c \). Output \( R^*_c \).

The cross-over operation is to create two new chromosomes (say, \( R_1 \) and \( R_4 \)) from two existing ones (say, \( R_1 \) and \( R_2 \)). Let each gene-segment \( i \) in \( R_1 \) and \( R_2 \) be respectively represented by \( r_{1i}^t \) and \( r_{2i}^t \). We proposed a one-point cross-over operation (Binh & Lan, 2007) on gene-segments \( r_{1i}^t \) and \( r_{2i}^t \) to create two new ones \( r_{3i}^t \) and \( r_{4i}^t \), which in turn could yield two new chromosomes:

\[
R_3 = [r_{1i}^t, r_{2i}^t, \ldots, r_{(i-1)i}^t, r_{3i}^t, r_{(i+1)i}^t, \ldots, r_{ni}^t], \quad 1 \leq i < n
\]

The one-point cross-over operation on a gene-segment is briefly introduced. For two gene-segments (i.e., \( r_{1i}^t \) and \( r_{2i}^t \)), we randomly choose a free gene, swap their gene values, and modify another gene values in order to ensure meeting the constraint \( a_t + b_t + c_t + d_t = 1 \). Consider an example, where the 2nd gene (a free one) is chosen as the cross-over point for mixing \( r_{1i}^t = (a_1, b_1, c_1, d_1) \) and \( r_{2i}^t = (a_2, b_2, c_2, d_2) \). By the swap and modification operations, we would obtain \( r_{3i}^t = (a_1, b_2, c_1, 1 - a_1 - b_2 - c_1) \) and \( r_{4i}^t = (a_2, b_1, c_2, 1 - a_2 - b_1 - c_2) \).

In the mutation operation, a new chromosome (say, \( R_5 \)) is created by an existing one (say, \( R_4 \)). The mutation algorithm creates \( R_5 \) by modifying a particular gene-segment in \( R_4 \). The modified gene-segment is randomly chosen. While being selected, two of its free genes are randomly chosen and their gene values are...
swapped. For example, if gene-segment $i$ is chosen for modification; and the 2nd and 4th genes are chosen to swap for 
$r_{t1} = (a_0, b_1, c_2, d_3)$, then $r_{t2} = (a_1, d_1, c_1, b_3)$, which in turn yield a new chromosome $r_2 = [r_{t2}, r_{t3}, \ldots, r_{t7}]$ from $r_1 = [r_{t1}, r_{t2}, \ldots, r_{t7}]$. Notice that only products in $Q_0$ are eligible for applying the mutation operation.

Two termination conditions are defined for the GA. First, the best solution in $P_t$ has not been changed for over a certain period (say, $T_k$ iterations). Second, population $P_t$ has evolved over a certain number of iterations; that is, $t$ has reached its predefined upper bound ($T_u$).

7. Experiments

Numeric experiments are carried out to compare the performance of our three proposed methods against the one proposed by Wu et al. (2008). The one proposed by Wu et al. (2008) is called $LP_0 – GA_0$. The three we propose are respectively called $LP_1 – GA_1$, $LP_2 – GA_2$, and $LP_3 – GA_3$. A personal computer equipped with Pentium (R) Dual CPU 3.4 GHz and 1 GB RAM is used in the experiments.

In the experiments, the data for machines, product routes and operation times are adapted from a data set provided by a semiconductor company. Each of the two fabs involves 60 workstations. Fab_1 involves 292 machines and Fab_2 involves 352 machines. The MTBF (mean time between failure) and MTTR (mean time to repair) of each machine is available, exponentially distributed.

Three scenarios are considered in the experiments. Scenario 1 involves three products (Table 1); Scenario 2 involves six products (Table 2); Scenario 3 involves nine products (Table 3). In the genetic algorithms, we set $T_k = 10,000$ and $T_u = 500$. $P_0 = 1000$ and $P_m = 0.1$. The target cycle time is $C_{T_0} = 40,000$ min or 27.7 days.

Table 4 compares the four methods in terms of the two fabs' aggregate throughput. Of the three proposed methods, $LP_1 – GA_1$ appears to be the best one, in particular in Scenario 3—only 2.48% less than $LP_3 – GA_3$ in throughput. However, the computation time required by $LP_2 – GA_2$ is greatly reduced. From Table 5, in dealing with Scenario 3, $LP_0 – GA_0$ requires 46.578 s (about 13 h), while $LP_2 – GA_2$ requires only 2197 s (about 35 min). In practice, taking half a day in computation is generally not acceptable to practitioners. Therefore, $LP_2 – GA_2$ appears to be a useful decision aid in solving cross-fab route planning problems.

Table 5 shows the two components of computation times required in Scenario 3. Table 6 indicates that the reduction in computation time is substantially due to the enhancement in LP. In the LP module, $LP_3 – GA_3$ takes 42.900 s (about 12 h) while $LP_2 – GA_2$ requires only 110 s (about 2 min).

The reasons why $LP_2 – GA_2$ outperforms the other two proposed methods, in terms of solution quality, are analyzed below. In $LP_1 – GA_1$, products in $Q_s$ are high-level in terms of bottleneck workload. This implies that these products are higher in product mix ratios. This leads to a higher eligible range for each route ratio in $Q_s$. In turn, the GA solution space of route ratios would become much larger. Under the same GA terminating conditions, the solution obtained by $LP_1 – GA_1$ may not be as good as that obtained by $LP_2 – GA_2$.

By contrast, in $LP_3 – GA_3$, products in $Q_s$ are low-level in terms of bottleneck workload; that is, products are generally lower in product mix ratios. This leads to a lower eligible range for each route ratio in $Q_s$. In turn, the space for improving the solution quality is also reduced. Therefore, $LP_2 – GA_2$ would outperform $LP_3 – GA_3$.

8. Conclusion

This paper presents an efficient approach to solve cross-fab route planning problems for semiconductor wafer manufacturing. In the problem, each product has four possible production routes,
which are defined by a cut-off point. We need to determine the cut-off point and the ratio range for each product in order to maximize the throughput subject a cycle time constraint.

A prior study has proposed a method (called LP\(_{p-GA_0}\)) to solve the problem, yet it is computationally extensive in dealing with large scale cases. In this paper, we enhanced the prior method and proposed three efficient methods (called LP\(_{1-GA_1}\), LP\(_{2-GA_1}\), and LP\(_{3-GA_1}\)). Numerical experiments indicate that the three enhanced methods can significantly reduce the required computation time. Of the three enhanced methods, LP\(_{2-GA_1}\) outperforms the other two in terms of solution quality, in dealing with large scale cases.

Some extensions of this research are being considered. The first extension is the route planning for a multiple-fab production system—for example, three or more fabs share the capacity in production. The second extension is the route planning for a scenario with higher flexibility in production routes—for example, each product could have two or more cut-off points and in turn have more than four routes.

Appendix 1

**Procedure Route_Planning**

**Step 1:**
- **Input** \((L, U)\)
- **Input** \((Q_1, Q_s)\)

**Step 2:**
- Call **Route_Planning_for_Given_Throughput** \((L, U, Q_1, Q_s)\)

**Step 3:**
- **Output** \(Z^*\), \(\Pi^*_l\), \(R^*_l\)

Appendix 2

**Procedure Route_Planning_for_Given_Throughput** \((L, U, Q_1, Q_s)\)

**Initialization** */ set initial range of throughput*

\[ i = 1, /* i is iteration numbers */ L_i = L, U_i = U \]

**While** \[ \{i = 1 or \frac{V_i - V_{i-1}}{V_{i-1}} > \epsilon \} /* \epsilon is a small value, e.g., 0.2% */

**Step 1:** Determine the two test points for the throughput interval \(I_i\)

\[ V_1 = (U_i + L_i)/4 \]
\[ V_2 = (3(U_i + L_i))/4 \]

**Step 2:** Evaluate and record the performance of the two test points

**Call** **Performance_Evaluation** \((V_1, Q_1, Q_s)\)

- \(P_1 = \text{Pass}_\text{Check}(V_1)\) /* Check if \(V_1\) is too large*/
- \(\Pi_1 = \text{Optimal}_\text{Cutoff}(V_1)\)
- \(Z_1 = \text{Optimal}_\text{Objective}_\text{Value}(V_1, \Pi_1)\)

**Call** **Performance_Evaluation** \((V_2, Q_1, Q_s)\)

- \(P_2 = \text{Pass}_\text{Check}(V_2)\) /* Check if \(V_2\) is too large*/
- \(\Pi_2 = \text{Optimal}_\text{Cutoff}(V_2)\)
- \(Z_2 = \text{Optimal}_\text{Objective}_\text{Value}(V_2, \Pi_2)\)

**Step 3:** Update the throughput interval for search
- If \((P_2 = \text{“Pass”})\) then \(L_{i+1} = (U_i + L_i)/2\), \(U_{i+1} = U_i\), \(k = 2\)
- If \((P_1 = \text{“Pass”})\) and \((P_2 = \text{“Fail”})\) then \(L_{i+1} = L_i\), \(U_{i+1} = (U_i + L_i)/2\), \(k = 1\)
- If \((P_1 = \text{“Fail”})\) and \((P_2 = \text{“Fail”})\) then \(L_{i+1} = L_i\), \(U_{i+1} = (U_i + L_i)/4\), \(k = 0\)

\[ i = i + 1 \]

**Endwhile**

- If \(k = 0\), Stop */ User warning: the input value of \(L\) is too large */
- Else \(Z^2 = Z_1, \Pi^*_l = \Pi_1, R^*_l = R_k\)

**Return** \(Z^*, \Pi^*_l, R^*_l\)

**Appendix 3**

**Procedure Performance_Evaluation** \((V_1, Q_1, Q_s)\)

**Assumption:** \(Q_1\) has \(n\) products, and the number of operations for product \(k\) is \(O_k\)

**Initialization**
- \(j = 1, /* iteration numbers */\)
- For each product \(k\), set its initial interval for search.
- \(L_k = 0, U_k = O_k, 1 \leq k \leq n\)
- \(I_k = [L_k, U_k], 1 \leq k \leq n\)

**Identify the longest route */ for terminating the following While loop */

- \(h = \text{ArgMax}_ {1 \leq k \leq n} O_k\)

**While** \((j = 1 or (m_{2h} - m_{1h}) \leq 1)\)

**Step 1:** Determine the two cut-off points for each segment \(I_k\)
- \(m_{1k} = ([U_k + L_k]/4), 1 \leq k \leq n\)
- \(m_{2k} = ([3(U_k + L_k))/4], 1 \leq k \leq n\)

**Step 2:** Generate all possible combinations of cut-off points \(S_j = \{\{\Pi_1\Pi_2 = (\pi_1, \ldots, \pi_n)\}, \quad \pi_k = m_{1k} \quad \text{or} \quad \pi_k = m_{2k}\}\)

**Step 3:** Identify the best combination of cut-off points from \(S_j\)

- **Set** \(H_1 = \phi, H_2 = \phi\)
- **For each** \(H_i \in S_j\)

**Call** **LP_Module** \((V_1, \Pi_1, Q_s)\)

- If \((\text{Pass}_\text{Check} = \text{“Pass”})\), put \(Z(V_1, H_1)\) in \(H_1\) and \(R(V_1, H_1)\) in \(H_2\)

**Endfor**

**Step 4:** Check if there exist a solution in \(S_j\)

- If \((H_1 \neq \phi)\), then \(H' = \text{ArgMin}_{\{\Pi_1\Pi_2\}} R(V_1, H')\) and \(R' = R(V_1, H')\)

- If \((H_1 = \phi)\), then \(\text{Pass}_\text{Check} = \text{“Fail”} \), **Return**

**Step 5:** Update the interval for each product \(k\)
- If \((\pi_k = m_{1k})\) then \(L_{j+1} = L_k, U_{j+1} = ([U_k + L_k]/2), 1 \leq k \leq n\)
- If \((\pi_k = m_{2k})\) then \(L_{j+1} = ([U_k + L_k]/2), U_{j+1} = U_k, 1 \leq k \leq n\)

\(j = j + 1 \)

**Endwhile**

- **Optimal_Cutoff** \((V) = H'\)
- **Optimal_Route_Ratio** \((V) = R'\)
- **Optimal_Objective_Value** \((V) = Z(V, H')\)

**Pass_Check** \((V) = \text{Pass}_\text{Check}\)

**Return**

**References**


Kim, C. O., Beak, J. G., & Jun, J. (2005). A machine cell formation algorithm for simultaneously minimizing machine workload imbalances and inter-cell part...