行政院國家科學委員會補助專題研究計畫成果報告

※※※※※※※※※※※※※※※※※※※※※※※※※
※　　　　　　　　　　　　　　　　　　　　　　　※
※　具實用性及交期導向之 IC 封裝廠排程演算法則　※
※　　　　　　　之設計與應用　　　　　　　　　　※
※　　　　　　　　　　　　　　　　　　　　　　　※
※※※※※※※※※※※※※※※※※※※※※※※※※

計畫類別：☑個別型計畫　　□整合型計畫

計畫編號：NSC　90－2218－E－009－019－

執行期間：90 年 08 月 01 日至 91 年 07 月 31 日

計畫主持人：彭文理教授

計畫共同主持人：鍾淑馨教授

計畫參與人員： 鄭博化

羅湘君

莊佳蒨

本成果報告包括以下應繳交之附件：

□赴國外出差或研習心得報告一份

□ 赴大陸地區出差或研習心得報告一份

□ 出席國際學術會議心得報告及發表之論文各一份

□國際合作研究計畫國外研究報告書一份

執行單位：國立交通大學工業工程管理學系

中　華　民　國　　九十一　年　九　月　十八　日

行政院國家科學委員會專題研究計畫成果報告

計畫編號：NSC 90-2218-E-009-19-

執行期限：90 年 08 月 01 日至 91 年 07 月 31 日

主持人：彭文理　交通大學工業工程與管理系

共同主持人：鍾淑馨　交通大學工業工程與管理系

計畫參與人員：鄭博化 羅湘君 莊佳蒨　交通大學工業工程與管理系

1. Background and Motivation

The major four sections for the IC manufacturing include the wafer fabrication, the wafer probing (wafer sorting), the IC packaging, and the final testing. And on time delivery is the major objective of IC packaging factories. Due to the high cost of the manufacturing by using the ceramic process, most of the IC packaging factories use plastic packaging. The manufacturing process of plastic packaging generally contains the following eleven process stages, (1) tapping, (2) lapping (wafer back grinding), (3) die sawing, (4) die bonding, (5) wire bonding, (6) molding, (7) marking, (8) plating, (9) de-flash and trimming, (10) forming, and (11) final testing (Manzione (1990)).

In the IC packaging factory, a group of machines are arranged in parallel at each stage of the process, and jobs in hundreds of product types are to be processed. The processing times of each job at different process stage may not be the same but predetermined. Each job must be processed on any of the parallel machines at each process stage. Therefore, the manufacturing system in the IC packaging factory can be considered as a flexible flow shop. The IC packaging scheduling problem (ICPSP) we investigate is a variation of the flexible flow-shop problem, and a generalization of the parallel-machine scheduling problem.

In real situations, the ICPSP involves constraints on job clusters, job-cluster dependent processing time, due dates, machine capacity, and sequentially dependent setup time at each of the multiple process stages. Therefore, the ICPSP is considerably more difficult to solve than the classical parallel-machine scheduling problem investigated by Ho and Chang (1995), Gabrel (1995), Schutten and Leussink (1996), Cheng and Gen (1997), Ruiz-Torres et al. (1997), Park and Kim (1997), and Lee and Pinedo (1997). The development of an efficient scheduling system is essential in the IC packaging industry, as it improves the efficiency of the production with dynamic orders, and tight due dates restrictions.

2. Purpose

In the research, we formulate the ICPSP as an IP problem to minimize the total machine workload. The IP model considers the due date restrictions, which includes the processing time and the setup time in the capacity constraints, thus reflects the real situations more accurately than those considered by Salvador (1973), Arthanari (1974), Wittrock (1988), and Sriskandarajah and Sethi (1989). In addition, we propose two efficient solution procedures to solve the

ICPSP approximately, with efficient job schedules that minimize the total machine workload. For most factories, wafer sawing (die sawing), die mounting, and wire bonding are three most critical manufacturing stages among all, and the machine numbers at those stages are enormous. Scheduling those machines under the production environment with many factors mentioned above, has become a rather difficult task for the IC packaging factories. For most cases, the due dates for the jobs are usually settled between the customers and the factories. Therefore, some tardiness (lateness) must be considered to ensure the job completion of the orders. Minimizing the total workload allows the process to fully utilize the system capacity, thus increases the overall productions.

3. Researching Theory

3.1 Framework

First, let the manufacturing environment consists of a set of processing stages, and all jobs must be processed through the P stages and completes the necessary operations within every stage. We also define M_p = $\{ m_{p1}, m_{p2}, ..., m_{pK} \}$ as the machine group at the p-th stage and containing a set of K_P identical machines. We note that the cluster R_0 includes $J_0 = J_{01} + J_{02} + \cdots + J_{0P}$ jobs in each process stage. Because the cluster R_0 is to denote the idle status of the K_p parallel machines of each stage, the number of the cluster R_0 in each stage are equal to the number of the parallel machines in each stage. Let $W_p = \{ W_{p1}, W_{p2}, ..., W_{pK} \}$ be the predetermined machine capacity at the p-th stage expressed in terms of processing time units. And then, we define R = $\{R_0, R_1, R_2, ..., R_I\}$ as the I + 1 clusters of jobs to be processed with each job cluster $R_i = \{r_{ij} \mid j = 1, 2, ..., J_i\}$ containing J_i jobs. Since each job r_{ij} must be processed through a set of manufacturing process consists of p-stage, $r_{ij} = \{r_{ijp} \mid p = 1, 2, P\}$ denote the job r_{ijp} processed on the p-th stage. Let n_{ij} be the lot size (number of dies) of job r_{ij}.

p_{ip} : the unit processing time for each die for each job r_{ijp} in cluster R_i ($r_{ijp} \in R_i$) on machine m_{pk},

$s_{ii'p}$: the sequentially dependent setup time between any two consecutive jobs r_{ijp} $(\in R_i)$ and $r_{i'j'p}$ $(\in R_{i'})$ from different job clusters on each machine m_{pk}.

x_{ijpk} : the variable indicating whether the job r_{ijp} is scheduled on machine m_{pk} of the p-th stage, with $x_{ijpk} = 1$ if job r_{ijp} is scheduled to be processed on machine m_{pk} of the p-th stage, and $x_{ijpk} = 0$ otherwise.

t_{ijpk} : the starting time for job r_{ijp} to be processed on machine m_{pk} of the p-th stage, where t_{ijpk} is between the time windows (b_{ij1}, e_{ijP}) of job r_{ij}.

(b_{ij1}, e_{ijP}) : the service time windows,

b_{ij1} : the earliest starting time to process job r_{ij1} of the first process

e_{ijP} : the latest starting time to process job r_{ijP} of the last process. $e_{ijP} = d_{ij} - n_{ij} p_{iP}$.

Let $y_{iji'j'pk}$ be the precedence variable, where $y_{iji'j'pk}$ should be set to 1 if the two jobs r_{ijp} and $r_{i'j'p}$ are scheduled on machine m_{pk} and job r_{ijp} precedes job $r_{i'j'p}$ (not necessarily directly), and where $y_{iji'j'pk}$ = 0 otherwise. Further, let $z_{iji'j'pk}$ be the direct-precedence variable, where $z_{iji'j'pk}$ should be set to 1 if the two jobs r_{ijp} and $r_{i'j'p}$ scheduled on machine m_{pk} and job r_{ijp} precedes job $r_{i'j'p}$ directly, and where $z_{iji'j'pk}$ = 0 otherwise.

3.2 Integer Programming Formulation

To find a schedule for the jobs which minimizes the total machine workload without violating the machine capacity and the service time windows constraints, we consider the following integer programming model shown by Table 1.

3.3 Multi-Stage Sequential Savings Algorithm

The MSSS algorithm for the ICPSP is essentially based on the well-known savings procedure of Clark and Wright (1964) for the vehicle routing problem, with some modifications. Each process stage, therefore, can be treated as a parallel-machine scheduling problem. Consequently, one solution strategy we may take is to solve each single-stage ICPSP problem sequentially to obtain the stage solutions, then combine them into a complete ICPSP solution.

The MSSS algorithm is proceeding as follows. First, apply the sequential-savings procedure to obtain stage 1 solution. We then use the job completion times obtained from stage 1 solution as the ready times b_{ijp} for the jobs to be processed at stage 2, and solve the ICPSP for stage 2. Repeat this step until we obtain all the P stage solutions. All the P stage solutions, are then combined to form a complete ICPSP solution. Variations of the MSSS algorithm may be considered by setting various due dates for each stage sequentially, then solve for the resulting single stage ICPSP to generating multiple solutions. The best among all combined solutions then is selected as the solution for the ICPSP.

The Sequential-Savings algorithm, initially, calculates the savings of all pairs of jobs and creates a list by sorting the savings in descending order of their magnitudes. The algorithm then selects the first feasible pair of jobs from the top of the list to start a new schedule (initialization of the first schedule). We note that a selected pair of jobs is feasible and will be added to the machine schedule if it does not violate the machine capacity constraints and the starting time windows constraints. Starting from the top of the savings list, the Sequential-Savings algorithm expands the schedule by finding the first feasible pair of jobs on the list then adding it to either one of the two ends of the schedule. If the current schedule cannot be expanded, choose the first feasible pair of jobs from the top of the list to start another new schedule. Repeat such steps until all jobs are scheduled.

The savings, $SA_{iji'j'p}$, which we considered in the MSSS algorithm is defined as $SA_{iji'j'p} = s_{\mathrm{U}ip} + s_{\mathrm{U}i'p} - s_{ii'p}$, for all pairs of jobs r_{ijp} and $r_{i'j'p}$, where $s_{ii'p}$ represents the setup time between any two consecutive jobs $r_{ijp} (\in R_i)$ and $r_{i'j'p} (\in R_{i'})$ from different job clusters on each machine m_{k_p} , the notation U denotes the machine is in idle status, and $s_{\mathrm{U}ip}$ ($s_{\mathrm{U}i'p}$) represents the setup time to prepare an idle machine to process job r_{ijp} ($r_{i'j'p}$) . We write a C^{++} program to proceed with the steps of the

Sequential-Savings algorithm. Considering the short run time it takes, and the single solution obtained, the MSSS algorithm can effectively solve large-scale ICPSP, and is considered efficient.

3.4 Multi-Stage Parallel Insertion Algorithm

The multi-stage parallel-insertion algorithm (MSPI) can effectively handle large-scale problems. The MSPI algorithm for the ICPSP is essentially based on the parallel-insertion procedure presented by Potvin and Rousseau's (1993) for the vehicle routing problem with time windows, with some modifications. Note that the IC packaging process consists of multiple manufacturing stages, each process stage therefore can be treated as a parallel-machine scheduling problem. Consequently, one solution strategy we may take is to solve each single-stage ICPSP problem sequentially to obtain the stage solutions, then combine them into a complete ICPSP solution.

The MSPI algorithm is proceeding as follows. First, apply the parallel-insertion procedure to obtain stage 1 solution. We then use the job completion times obtained from stage 1 solution as the ready times b_{ijp} for the jobs to be processed at stage 2, and solve the ICPSP for stage 2. Repeat this step until we obtain solutions for all K stages. All K stage solutions, are then combined to form a complete ICPSP solution. Variations of the MSPI algorithm may be considered by setting various possible due dates for each stage sequentially, then solve for the resulting single stage ICPSP to generating multiple solutions. The best among all combined solutions then is selected as the solution for the ICPSP.

At the initialization step, the parallel-insertion procedure constructs a set of machine schedules simultaneously. The procedure uses a generalized regret measure over all schedules to select the best unscheduled job, which can foresee the difficulty of inserting jobs into machine schedules. Let PS_{pk} be the partial schedule of machine m_{pk} at process stage p, where

$$PS_{pk} = (u_{pk0}, \dots, u_{pk(n-1)}, u_{pkn}, \dots, u_{pkL})$$

$$u_{pk0} = u_{pkL} = 0, \quad k = 1, \dots, K_p, \quad p = 1, \dots, P$$

where u_{pkn} is the n-th job scheduled on machine m_{pk} at the process stage p, and u_{pk0} and u_{pkL} represent pseudo jobs. For each unscheduled job r_{ijp} assigned to machine m_{pk}, we first compute its insertion cost on each position of the partial schedule of each machine at the process stage p.

At process stage p, $ë_{pk}\ (u_{pk(n-1)}, r_{ijp}, u_{pkn})$ represents the additional setup time occurred if job r_{ijp} is inserted between the (n - 1)th and n-th positions of the partial schedule PS_{pk}. In some cases, job insertion may cause the postponement of starting processing time of jobs already on the partial schedule. If the postponement is against the starting service time windows constraint, $ë_{pk}\ (u_{pk(n-1)}, r_{ijp}, u_{pkn})$ is set to be an arbitrary large value.

$$ë_{pk}(u_{pk(n-1)}, r_{ijp}, u_{pkn})$$

$$=$$

$$s_{I(u_{pk(n-1)})ip} + s_{iI(u_{pkn})p} - s_{I(u_{pk(n-1)})I(u_{pkn}}$$

where $u_{pk(n-1)}$ and u_{pkn} are two consecutive jobs on partial schedule PS_{pk} before job r_{ijp} is inserted, $I(u_{pkn})$ is the function that returns the product type of the job being scheduled on the n-th position of partial schedule PS_{pk}.

Let $ë^{*}_{pk}\ (u_{pk(n^{*}-1)}, r_{ijp}, u_{pkn^{*}})$ denote the lowest insertion cost due to the insertion of job r_{ijp} into the partial schedule PS_{pk}.

$$ë^{*}_{pk}(u_{pk(n^{*}-1)},\ r_{ijp},\ u_{pkn^{*}}) = \min_{n=1,\dots,l} [ë_{pk}(u_{pk(n-1)},\ r_{ijp},\ u_{pkn})]$$

However, we may not choose next inserted job r_{ijp} with lowest $ë^{*}_{pk}\ (r_{ijp})$, since insertion problem may occur for jobs with larger insertion cost. To quantify the future insertion difficulties for a job on each machine, the generalized regret measure $\sigma(r_{ijp})$ of job r_{ijp} is defined. The regret measure looks ahead what can be lost later, if a given job is not immediately inserted into its best alternative machine at the process stage p, which summaries the differences of insertion cost between the best alternative machine and all other alternative machines. Hence, unscheduled jobs with larger regret value must be inserted in higher priority, since these jobs are more difficult to find feasible insertion position among all machines. In the parallel insertion algorithm, instead of using $ë^{*}_{pk}(r_{ijp})$, the candidate inserted job r^{*}_{ijp} is selected with largest regret value $\sigma(r_{ijp})$.

$$ó(r^{*}_{ijp}) = \max_{r_{ijp}}[\sigma(r_{ijp})]$$

$$\sigma(r_{ijp}) = \sum_{pk \neq pk'} [ë^{*}_{pk}(u_{pk(n^{*}-1)}, r_{ijp}, u_{pkn^{*}}) - ë^{*}_{pk'}(u_{pk'(n^{*}-1)}, r$$

where

$$\ddot{e}^*_{pk'}(u_{pk'(n^*-1)}, r_{ijp}, u_{pk'n^*}) = \min_{k=1,\ldots,K_p}[\ddot{e}^*_{pk}(u_{pk(n^*-1)}, r_{ijp}, u_{pkn^*})]$$

and insert r^*_{ijp} between $u_{pk'(n^*-1)}$ and $u_{pk'n^*}$. At each process stage, the steps of algorithm are described in the following:

(Step 1) At each process stage, obtain the partial schedules for all machines using the largest criterion to select a jobs r_{ijp}, which causes an idle machine m_{pk} to spend the largest setup time to prepare for the processing of job r_{ijp}.

(Step 2) The following three sub-steps are proceeded to execute the scheduling procedures until all the jobs are scheduled.

(a) For each unscheduled lot, first compute its best feasible insertion position by $\ddot{e}^*_{pk}(u_{pk(n^*-1)}, r_{ijp}, u_{pkn^*})$ at each machine's partial schedule PS_{pk}.

(b) Compute the regret value $\sigma(r_{ijp})$ for each job. Choose the next inserted job r^*_{ijp} with largest $\sigma(r_{ijp})$ among all unscheduled jobs.

(c) The best lot r^*_{ijp} is inserted into the lowest insertion cost position of the machine determined by $\ddot{e}^*_{pk}(r^*_{ijp})$.

We implement the MSPI algorithm using the C^{++} programming language to execute the Parallel-Insertion procedure at each of the P stages.

4. Achievements

In this study, we formulated ICPSP as an integer programming model the presented an efficient solution procedure, called the Multi-Stage Parallel Insertion algorithm, to solve the ICPSP case, which minimizes the total machine workload.

Major achievements:

1. Develop the integer programming model for the ICPSP with the total machine workload minimized. And write C++ programming language code to generate the IP model.
2. Use integer programming software (Cplex) to illustrate the applicability of the linear integer programming model, and derive the optimal solution for ICPSP. Transfer the ICPSP the ICPSP into vehicle routing problem with time-window(VRPTW)network problem.
3. Develop multi-stage sequential savings algorithm (MSSS) to solve the ICPSP approximately and effectively handle large-scale problems.
4. Develop multi-stage parallel-insertion algorithm (MSPI) to solve the ICPSP approximately and effectively handle large-scale problems.

References

[1] Arthanari, T. S. (1974). On some problems of sequencing and grouping. PhD thesis, Indian Statistical Institute, Calcutta.

[2] Cheng, R. and Gen, M. (1997). Parallel machine scheduling problem using memetic algorithms. Computers and Industrial Engineering, 33(3-4), 761-764.

[3] Gabrel, V. (1995). Scheduling jobs within time windows on identical parallel machines: new model and algorithms. European Journal of Operational Research, 83, 320-329.

[4] Ho, J. C. and Chang, Y. L. (1995). Minimizing the number of tardy jobs for m parallel machines. European Journal of Operational Research, 84, 343-355.

[5] Lee, Y. H. and Pinedo, M. (1997). Scheduling jobs on parallel machines with sequence-dependent setup times. European Journal of Operational Research, 100, 464-474.

[6] Manzione, L. T. (1990). Plastic Packaging of Microelectronic Devices. Technical Report, AT&T Bell Laboratories.

[7] Ovacik, I. M. and Uzsoy, R. (1996). Decomposition methods for scheduling semiconductor testing facilities. The International Journal of Flexible Manufacturing Systems, 8, 357-388.

[8] Park, M. W. and Kim, Y. D. (1997). Search heuristics for a parallel machine scheduling problem with ready time and due dates. Computers and Industrial Engineering, 33(3-4), 793-796.

[9] Potvin, J. Y. and Rousseau, J. M. (1993). A Parallel route building algorithm for the vehicle routing and scheduling problem with time windows. European Journal of Operational Research, 66, 19-26.

[10] Salvador, M. S. (1973). A solution of a special class of flow-shop scheduling problems. Proceedings of the Symposium on the Theory of Scheduling and its Applications, 83-91, Springer-Verlag, Berlin.

[11] Schutten, J. M. J. and Leussink, R. A. M. (1996). Parallel machine scheduling with release dates, due dates and family setup times. International Journal of Production Economics, 46-47, 119-125.

[12] Sriskandarajah, C. and Sethi, S. P. (1989). Scheduling algorithms for flexible flowshop: worst and average case performance. European Journal of Operational Research, 43, 143-160.

[13] Wittrock, R. J. (1988). An adaptable scheduling algorithm for flexible flow lines. Operations Research, 36, 445-453.

Table 1 The IP model for ICPSP

minimize
$$\sum_{l=1}^{L}\sum_{k=1}^{K}\{\sum_{i=0}^{I}\sum_{j=1}^{J_i} x_{ijpk}\, n_{ij}\, p_{ip} + \sum_{i=0}^{I}\sum_{j=1}^{J_i}(\sum_{i'=0}^{I}\sum_{j'=1}^{J_{i'}} z_{iji'j'pk}\, s_{ii'p})\}$$

subject to

$$\sum_{k=1}^{K_p} x_{ijpk} = 1, \quad \text{for all } i, j, p \tag{1}$$

$$\sum_{j=1}^{J_0} x_{0jpk} = 1, \quad \text{for all } p,k \tag{2}$$

$$\sum_{i=0}^{I}\sum_{j=1}^{J_i} x_{ijpk}\, n_{ij}\, p_{ip} + \sum_{i=0}^{I}\sum_{j=1}^{J_i}(\sum_{i'=0}^{I}\sum_{j'=1}^{J_{i'}} z_{iji'j'pk}\, s_{ii'p}) \le W_{pk}, \quad \text{for all } p,k \tag{3}$$

$$(y_{iji'j'pk} + y_{i'j'ijpk}) - Q_p(x_{ijpk} + x_{i'j'pk} - 2) \ge 1, \quad \text{for all } i, j, p,k \tag{4}$$

$$(y_{iji'j'pk} + y_{i'j'ijpk}) + Q_p(x_{ijpk} + x_{i'j'pk} - 2) \le 1, \quad \text{for all } i, j, p,k \tag{5}$$

$$(y_{iji'j'pk} + y_{i'j'ijpk}) - Q_p(x_{ijpk} + x_{i'j'pk}) \le 0, \quad \text{for all } i, j, p,k \tag{6}$$

$$(y_{iji'j'pk} + y_{i'j'ijpk}) - Q_p(x_{i'j'pk} - x_{ijpk} + 1) \le 0, \quad \text{for all } i, j, p,k \tag{7}$$

$$(y_{iji'j'pk} + y_{i'j'ijpk}) - Q_p(x_{ijpk} - x_{i'j'pk} + 1) \le 0, \quad \text{for all } i, j, p,k \tag{8}$$

$$y_{iji'j'pk} \ge z_{iji'j'pk}, \quad \text{for all } i, j, p,k \tag{9}$$

$$\sum_{i'=1}^{I}\sum_{j'=1}^{J_{i'}} z_{iji'j'pk} \le 1, \quad \text{for all } i,j,p,k \tag{10}$$

$$\sum_{i'=1}^{I}\sum_{j'=1}^{J_i} z_{i'j'ijpk} \le 1, \quad \text{for all } i,j,p,k \tag{11}$$

$$\sum_{i=0}^{I}\sum_{j=1}^{J_i} x_{ijpk} - \sum_{r_{ij} \ne r_{i'j'}} z_{iji'j'pk} = 1, \quad \text{for all } p,k \tag{12}$$

$$t_{ijpk} + n_{ij}p_{ip} + s_{ii'p} - t_{i'j'pk} + Q_p(y_{iji'j'pk} - 1) \le 0, \quad \text{for all } i, j, p, k \tag{13}$$

$$t_{ijpk} + n_{ij}p_{ip} + s_{ii'p} - t_{i'j'pk} + Q_p(y_{iji'j'pk} + z_{iji'j'pk} - 2) \le 0, \quad \text{for all } i, j, p, k \tag{14}$$

$$t_{i'j'(p-1)k'} + n_{i'j'}p_{i'(p-1)} - t_{i'j'pk} + Q_p(y_{iji'j'pk} + z_{iji'j'pk} - 2) \le 0, \quad \text{for } p = 2,3,\ldots,P \text{ and all } i, j, p, k \tag{15}$$

$$y_{iji^*j^*pk} + z_{iji^*j^*pk} - Q_p(y_{iji^*j^*pk} + z_{iji^*j^*pk} - 2) - Q_p(y_{iji'j'pk} - z_{iji'j'pk} - 1) \ge 2, \text{ for all } i, j, p, k \tag{16}$$

$$t_{ijpk} + n_{ij}p_{ip}x_{ijpk} \le t_{ij(p+1)k'} + G1_p(1 - x_{ijpk}) + G2_p(1 - x_{ij(p+1)k'}), \text{ for } i = 1,2,\ldots, I \text{ and } p = 1, 2, \ldots, P-1, \text{ and all } j, k \tag{17}$$

$$\sum_{p=1}^{P}\sum_{k=1}^{K_p} x_{ijpk} = P, \quad \text{for } i = 1,2,\ldots, I \text{ and all } j \tag{18}$$

$$t_{ijpk} \ge b_{ij1}x_{ijpk}, \quad \text{for all } i, j, p, k \tag{19}$$

$$t_{ijpk} \le e_{ijP}x_{ijpk}, \quad \text{for all } i, j, p, k \tag{20}$$

$$x_{ijpk} \in \{0, 1\}, \quad \text{for all } i, j, p,k \tag{21}$$

$$y_{iji'j'pk} \in \{0, 1\}, \quad \text{for all } i, j, p,k \tag{22}$$

$$z_{iji'j'pk} \in \{0, 1\}, \quad \text{for all } i, j, p,k \tag{23}$$