Scheduling Algorithms for Computer-Aided Line Balancing in Printed Circuit Board Assembly
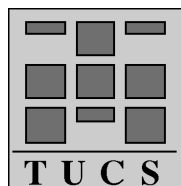
Timo Häyrinen
Mika Johnsson
Tommi Johtela
Jouni Smed
Olli Nevalainen

Abstract

Generalized Flexible Flow Line (GFFL) is a scheduling environment comprising several machine banks which the products visit in the same order but can skip some machine banks. The products are transported either in magazines or by a conveyor belt, and the change of the products demands a set-up operation of the machine. The time consumption of this operation depends on the current and the next product to be processed.

This paper describes several scheduling algorithms for GFFL problem. The overall structure of these algorithms is similar, consisting of machine allocation and sequencing phases. The algorithms have been implemented and integrated into an interactive production scheduling system for electronic assembly. Sample cases are used to illustrate the operation of the system in practice.

Keywords: flow line scheduling, approximate algorithms, interactive scheduling, production planning, printed circuit board assembly

TUCS Research Group
Algorithmics

1 Introduction

It is characteristic of many branches of advanced production technologies that the production program comprises a large number and several variants of different products. The concept of *Flexible Manufacturing Systems* (FMS) [14, 11] supports easy and cost-effective manufacturing of various products. The main idea of these systems is using the same set of flexible machines for processing all the products. A change from one product to another requires a *set-up* operation, the time of which depends on the current and next product to be manufactured. With a suitable production sequence we can decrease the total amount of the set-up times during which the machines stay idle. Furthermore, we can control the finishing times and the due dates by *scheduling* the jobs. In many cases the production planner is mainly concerned with the due dates, and the minimization of the set-up times may sometimes conflict with this goal. The construction of a feasible and, preferably, an efficient schedule is one of the most difficult tasks in production planning. It has been shown that the problem is too complicated to be solved accurately (even in theory) [5, 3]. Therefore, the problem is usually approached by the use of approximative algorithms.

There are two features that cause difficulties in the production environment. The manufacturing plant has usually multiple copies of operationally identical machine types organized as machine banks. The banks can also be considered as production phases. Each product has to pass these phases in a predefined order (*flow line processing*). Hence, the system may process several different types of products at a given time. The routing of the products affects the total production efficiency. The second difficulty originates from the fact that the production plan is subjected to due date constraints which imply the preference of feasible schedules of the jobs (i.e., *product batches*). Therefore, searching for an optimal schedule is not reasonable objective in applications of practical value.

There are three different approaches to the flow line scheduling. In the *algorithmic approach* the scheduling task is expressed as a mathematical optimization problem and is solved by an approximate algorithm [15, 9]. In the *interactive scheduling* the production designer uses computer simulation to evaluate different schedules [12]. The *hybrid approach* integrates both approaches [8]; the algorithms are used to produce a set of possible schedules which can be then evaluated and manipulated by the interactive scheduling tool.

In this paper we consider the hybrid approach. We concentrate on efficient algorithms and their integration to the interactive scheduler. Our examples are taken from printed circuit board (PCB) assembly[1]. The paper is organized as follows. In Section 2 we present the general framework for our scheduling problem by introducing the *generalized* flexible flow line scheduling. We also give the necessary notations used in the algorithms. In Section 3 we describe the scheduling algorithms and in Section 4 we give a practical example. Concluding remarks appear in Section 5.

2 Generalized Flexible Flow Lines

A *Generalized Flexible Flow Line* (GFFL) consists of m machines (possibly of several different *machine types*). The machines are grouped into n *phases*, each comprising one or more machines. Some of the machines in successive phases are organized as *logical* (flexible) or *physical* (conveyor belt) *lines*. The products pass the phases in the same order of phases. A given product is processed either by one of the machines of a phase or the phase may be skipped. Several copies of the same product are manufactured. The time to process a product on a machine depends on the combination of the product and machine (*processing time*) and the previous product on the machine (*set-up time*). Therefore, it is efficient to manufacture products of the same type in *batches*. A batch can also be splitted if it is considered to be too big (However, this must be done before the process begins).

The problem is called GFFL because of the fixed order of machine visits (-FL), multiple machines in a phase, the possibility of skipping a machine (-F-), the consideration of set-up times and the usage of magazines (G-). The problem can be expressed in the three-tuple notation (see [3]) as $FMPM/p_{ij}, d_i, \text{batch}/\min \sum T^2$ which stands for a Flow-Shop with m machines; jobs with no pre-emption (i.e., the processing of a job on a machine may not be interrupted); no precedence relations; all jobs are ready for processing; processing demands differ; deadlines; batches; and the optimization criterion is the minimization of the total sum of squared tardinesses of the batches. For simplicity we assume that the processing times are stochastic but the averages of the processing times are accurate enough for evaluating different *schedules*. The schedule determines the machine *allocation* and the *sequence* (ordering) of the jobs on each machine. Because the set-up times are product-dependent, we introduce the concept of product *families* [10]. Product families vary from phase to phase. For a given phase a product fam-

---

[1]This work has been done in co-operation with Nokia Display Products of Nokia Corporation, Finland as a part of a project on the optimization of the production plant.

ily consists of similar products so that the set-up time between the products is short within the same family.

Ammons *et al.* [1] categorize the strategies for set-up management as follows:

1. *Single set-up strategy* in which a group of machines is configured to produce a family of PCBs using a single set-up:

    (a) *Unique set-up strategy* in which the family contains only one product type (i.e., mass production).

    (b) *Family set-up strategy* in which the family comprises several product types.

2. *Multi set-up strategy* in which limited component staging capacity prohibits applying the single set-up strategy (see also [2]):

    (a) *Decompose and sequence* in which the family is divided into subfamilies which are then sequenced to minimize the incremental set-ups between subfamilies.

    (b) *Partition and repeat* in which the required components are partitioned into subsets restricted by machine capacity.

The concept of families is natural to many production environments; in *printed circuit board* (PCB) assembly there are usually several different variations of the same basic product. These variants may have only a small set of different components so that they can be processed with the same component set-up. Therefore, the set-up times are short when changing from one product to another of the same family.

Crama *et al.* [4] divide the problems of PCB production planning into five categories:

1. partitioning the set of board types into families,

2. partitioning the set of component locations for each board type,

3. determining the feeder set-up for each machine,

4. assigning a component placement sequence for each pair consisting of a machine and a board type, and

5. assigning a component retrieval plan for each pair of machine and board type.

We assume in this paper that the product family partitioning is given (or can be determined automatically from the product data by a simple analysis, for more details, see [13]).

We use the following notations:

n production phase, $n = 1, 2, \ldots, N$

m machine (index) in phase n, $m = 1, 2, \ldots, M_n$

j job (batch), $j = 1, 2, \ldots, J$

w_{nj} amount of work to be performed in phase n for product j

t_{nmjk} set-up time from job j to job k on machine m of phase n

f_{nj} the family (index) of job j in phase n

b_n the number of families in phase n

F_{ni} the set of products belonging to family i of phase n, i.e., $F_{ni} = \{j | f_{nj} = i, j \in \{1, 2, \ldots, J\}\}$, for $i = 1, 2, \ldots, b_n (\leq J)$

H_{nml} 1, if machine m of phase n and machine l of the phase $n + 1$ are connected by a fixed or logical conveyor belt; 0, otherwise.

The families form a proper partitioning of the products; i.e., for each $n(= 1, 2, \ldots, N)$

$$F_{ni} \subseteq \{1, 2, \ldots, J\}, \text{ for } i = 1, 2, \ldots, b_n$$

$$F_{nk} \cap F_{nl} = \varnothing, \text{ for } k \neq l \text{ and } k, l = 1, 2, \ldots, b_n$$

$$\bigcup_{i=1}^{b_n} F_{nl} = \{1, 2, \ldots, J\}$$

Note that some of the families may be singular. If two jobs k and l belong to the same family in phase n, they need not to be in the same family in a different phase n'; i.e. $f_{nk} = f_{nl}$ does not imply $f_{n'k} = f_{n'l}$.

The concept of family reflects on the set-up times so that

$$t_{nmkl} = \begin{cases} t'_{nm}, & \text{if } f_{nk} \neq f_{nl} \\ t''_{nm}, & \text{if } f_{nk} = f_{nl} \quad (t''_{nm} \ll t'_{nm}) \end{cases}$$

The above formulation of our scheduling problem simplifies somewhat the actual production process. To be more accurate we should consider the (somewhat) varying speeds of different machines in a phase. The products are transported between phases in so-called *magazines*. However, these two details can be omitted from the model without a risk of oversimplification.

There are several objectives that one should consider when constructing a good schedule for the jobs:

- keeping product families together,

- meeting the due dates,

- separating the allocation and sequencing phases,

- preserving the ordering of the completion times in the starting times of the next phase,

- work load balancing of the machines,

- minimizing the size of the internal storages, and

- minimizing the machine idle times.

These ideas and goals are contradictory in many cases, which makes the scheduling extremely complicated. The objective function in our scheduling problem is

$$C(S) = a \cdot \sum \text{squared tardiness}_j + b \cdot \sum \text{internal buffer size}_m$$
$$+ c \cdot \sum \text{internal waiting time}_j + d \cdot \sum \text{number of families}_j,$$

where the *tardiness* of a batch is the difference of the due date and the actual finishing time; the summation by m and j is over the machines and over the jobs, respectively. The multipliers a, b, c and d are required to emphasize different criteria. Usually the weight a is the most important. For an integer programming formulation of the GFFL, see [7].

3 Algorithms

In an *Interactive Production Scheduler* (IPS) [8] the production designer can trigger the use of one or more scheduling algorithms, see Fig. 1. The algorithms are fully integrated to the system so that no special input or output is necessary. The results can be evaluated and re-scheduled by the IPS directly.

Each of the algorithms accepts a *Production Plan* (PP) that gives the number of products and the due date for each batch j. The algorithms have access to the *Production Data* (PD) which include information about the machines (e.g., operation rate), short (inner-family) and long (inter-family) set-up times t''_{nm} and t'_{nm}, line organization etc. *Batch Data* (BD) describe the products by giving the number of machine operations for each production
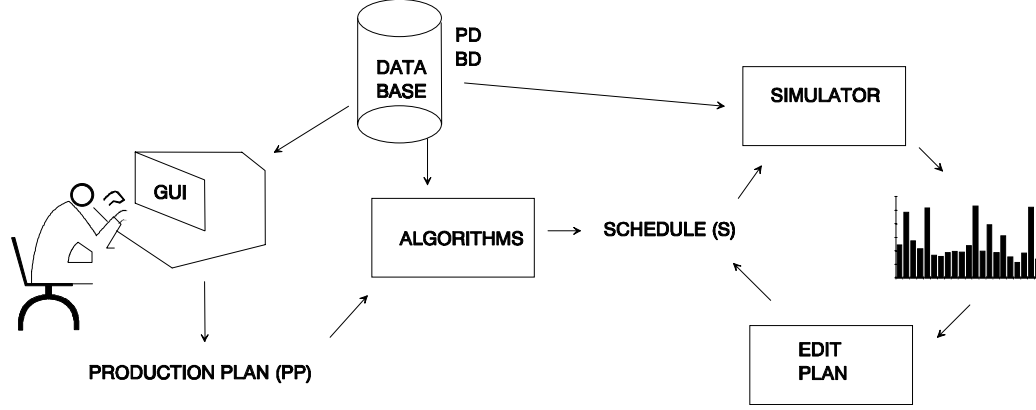
Figure 1: The IPS (Interactive Production Scheduler)

phase w_{nj}, the product family f_{nj}, etc. The PD and BD files are subjected to gradual changes as the machine installation and product spectrum develop. The algorithms produce a *Schedule* (S) which gives for all the machines of the installation a list of the product batch numbers in the order of manufacturing.

The scheduling algorithm consists of four steps (see Fig. 2):

1. Initial allocation of the batches to the machines

2. Improvement of the machine allocations

3. Initial sequencing of the batches

4. Improvement of the batch sequences

The scheduler includes three algorithms for allocation, five for improving the allocation, one for initial sequencing and two for improving sequencing. These are as follows:

1. Initial batch allocation to the machines
   a) allocation by batches
   p) allocation by families
   r) random allocation of batches

2. Improving the machine allocation
   l) local greedy improvement
   g) the globally best pair algorithm
   k) pairwise optimal, all pairs
   s) pairwise optimal, random pairs
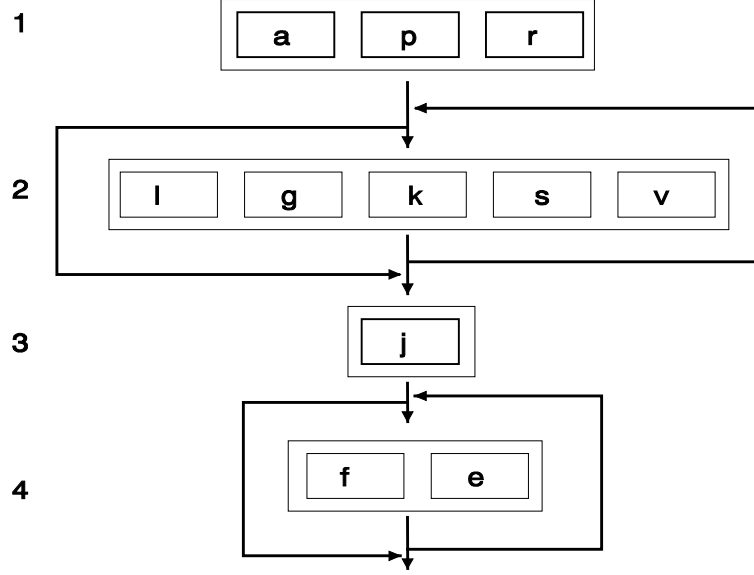   v) selection by function, pairwise optimal allocation

6

Figure 2: The main control flow of the scheduling algorithm

   3. j) Initial ordering of the batches

   4. Improving the schedule
      f) re-scheduling the whole families
      e) re-scheduling the batches of a family

The user can select any combination of these methods and repeat the improvement steps as many times as needed with the same or different method. This grants us, apart from repetitions, 30 different variants of the algorithm. The scheduling methods are identified by the above initial character notation (e.g., plje = allocation by families followed by the local greedy improvement, the initial ordering of the batches and the re-scheduling of the batches of a family).

Next, we discuss the four phases of the algorithm in more detail. Because the problem model is rather complicated, we give only an outline of the solutions. A full implementation of this algorithm is given in [6] and available from the authors. When describing the algorithms we introduce several new notations which are summarized in Appendix A.

3.1   Initial Batch Allocation to Machines

In the first phase the scheduler determines an initial allocation of the jobs to the machines. In this phase we aim at an initial situation which can then be tuned to a balanced machine allocation in the next phase.

3.1.1 Allocation by Batches

We have two conflicting goals in this allocation rule:

- equal work load of machines for each phase, and

- preserving the integrity of the families by allocating all products of a certain family to the same machine if possible.

To facilitate the allocation we introduce a parameter $\mathsf{fair\_share}_n$ for each phase. This parameter gives us the first approximation of the work load of each machine in phase n. It also gives the optimal allocation of the machines when the work loads of the machines are equal. The parameter restricts the addition of new batches disregarding the work load balancing among different machines of the phase. The work load of a particular set of batches is composed of two components: the total processing time and the sum of the set-up times. Thus, we calculate the limit for each phase $n = 1, 2, \ldots, N$ as follows

$$\mathsf{fair\_share}_n = \frac{1}{M_n} \sum_{j=1}^{J} w_{nj} + c_0 \cdot \min_{m=1,2,\ldots,M_n} \{t'_{nm}\} \tag{1}$$

The first term is the share of the actual processing time among the M_n machines. The second term considers the time usage of the major set-up operations. The coefficient c_0 is used for fine-tuning the algorithm; with it we can control how much work load imbalance we tolerate in order to keep families together.

To solve the conflict between work load balancing and family integrity we introduce another parameter called $\mathsf{share\_excess}$. With this parameter we will restrict the excess of work load in machines that process large families.

At a coarse level the allocation algorithm considers each phase n separately. We sort the jobs into a decreasing order by the processing demands of the phase, and process the jobs in this order (*Longest Processing Time First*). The actual allocation of the current job j is easy if a machine is found for which the next two conditions hold:

1. the machine already contains some jobs that belong to the same family as job j, and

2. the inequation $W \leq \mathsf{fair\_share}_n - w_{nj}$ holds, where W is the current work load of the machine.

These considerations give us four heuristic rules for allocating job j.

Rule 1 Among the machines fulfilling the conditions 1 and 2 choose the one for which the work load is minimal and allocate job j to it.

Rule 2 If condition 1 fails for each machine, allocate job j to the machine which has the lowest work load.

Rule 3 If condition 2 is violated, we choose the machine which fulfills condition 1. We apply this rule only when the work load of the selected machine is at most share_excess $\cdot$ fair_share$_n$ after the possible allocation. (By this we want to prevent the accumulation of a very high work load at some machines with large families.) Thus, we finally abandon the goal of keeping the families together.

Rule 4 If rule 3 is not applicable for any machine, job j is allocated to the machine with the minimal work load so far.

The allocation of the machines is restricted by the fact that certain machines in successive phases are interconnected. In that case the allocation of the latter machine is the same as the allocation of the first one.

Next, we outline an algorithm which implements the allocation rules 1–4.

Algorithm fair_share. The algorithm allocates jobs $j = 1, 2, \ldots, J$ to machines $m = 1, 2, \ldots, M_n$ in phases $n = 1, 2, \ldots, N$. Each job is allocated only to one machine. The input of the algorithm includes

w_{nj}	processing demands
t'_{nm}	long set-up time
t''_{nm}	short set-up time
f_{nj}	family indexes
H_{nml}	machine line information
fair_share$_n$	allocation limit
share_excess	allocation limit

The algorithm determines for each machine the set of jobs G_{nm} allocated to it. Variables W_{nm} and B_{nm} are used for storing the cumulative work load and the family index sets of the machines. S, T, R are auxiliary sets of machine indexes.

Step 1 (Process the machine phases) Perform the steps 2–6 for all phases $n = 1, 2, \ldots, N$.

Step 2 (Initialize the allocation for a phase) Let $G_{nm} \leftarrow \varnothing; B_{nm} \leftarrow \varnothing; W_{nm} \leftarrow 0$ for all $m = 1, 2, \ldots, M_n$. Sort the jobs into a decreasing order by the processing demand w_{nj}.

Step 3 (Process the jobs) Perform the steps 4 to 6 for each job j in the order determined at step 2.

Step 4 (Check the allocation conditions) Let S be the set of machines for which $f_{nj} \in B_{nm}$ (at least one other job of the same family has been allocated previously to the machine) and let T be the set of machines for which $W_{nm} \leq \mathsf{fair\_share}_n - w_{nj}$ (the $\mathsf{fair\_share}_n$ will not be overridden).

Step 5 (Select the machine)

Case 1 If $S \cap T \neq \varnothing$ then let m_0 be such that $W_{nm_0} = \min_{m \in S \cap T} \{W_{nm}\}$ and go to step 6.

Case 2 If $S = \varnothing$ then let m_0 be such that $W_{nm_0} = \min_{m=1,2,\ldots,M_n} \{W_{nm}\}$ and go to step 6.

Case 3 If $T = \varnothing$ let $R = \{m | W_{nm} + w_{nj} \leq \mathsf{share\_excess} \cdot \mathsf{fair\_share}_n; f_{nj} \in B_{nm}; m \in S\}$ (i.e., the set of machines with family f_{nj} and low work load). If $R \neq \varnothing$ then let m_0 be the machine for which $W_{nm_0} = \min_{m \in R} \{W_{nm}\}$ and go to step 6.

Case 4 Otherwise, let m_0 be such that $W_{nm_0} = \min_{m=1,2,\ldots,M_n} \{W_{nm}\}$.

Step 6 (Allocate)

$G_{nm_0} \leftarrow G_{nm_0} \cup \{j\}$ (set of jobs)

$W_{nm_0} \leftarrow W_{nm_0} + w_{nj}$ (work load)

$B_{nm_0} \leftarrow B_{nm_0} \cup f_{nj}$ (set of families)

Note that we have intentionally omitted the line machines. If we assume that their jobs are fixed by the first machine, we can add to the initialization step 2 the following test:

If $H_{n-1,l,m} = 1$ for some l then $G_{nm} \leftarrow G_{n-1,l}$, $W_{nm} \leftarrow \infty$ and $B_{nm} = B_{n-1,l}$.

Thus, we fill up the machine with the jobs of the previous machine and do not consider it further; the latter line machines accept their jobs from the line only.

Fig. 3 demonstrates the principle of the $\mathsf{fair\_share}$ allocation. Note that the job 1 should be allocated to the machine 1 which contains a job in the same family but the $\mathsf{fair\_share}$ limit would then be violated. Therefore, machine 2 is selected due to its minimal work load.

a) Jobs in the allocation order
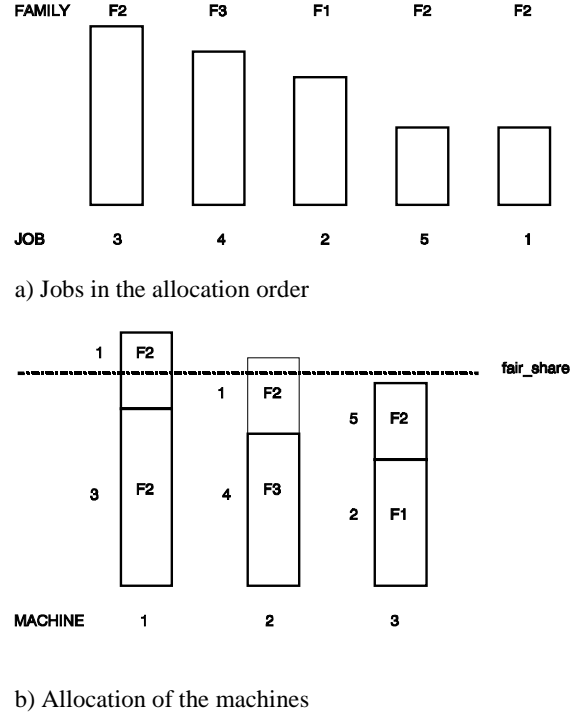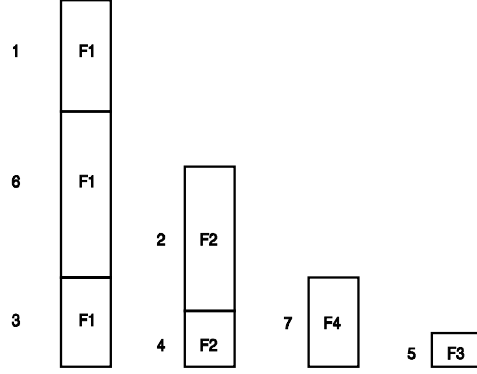

b) Allocation of the machines

Figure 3: Initial allocation by fair_share technique
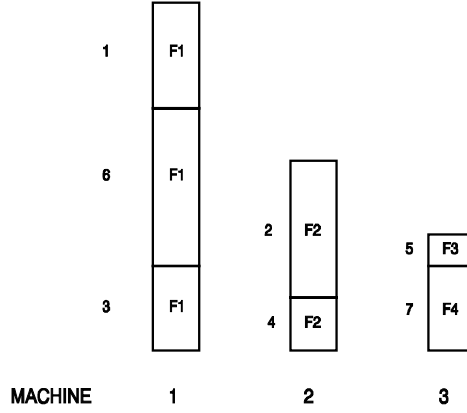
3.1.2 Allocation by Families

In this allocation method we consider the families in the *Longest Processing Time First* order (LPTF). A family is always allocated to the machine with the lowest work load. The parameter fair_share$_n$ is not considered. The method preserves the integrity of the families but may produce very skew initial allocations, see Fig. 4 In spite of this problem, the allocation method works well in scheduling. Therefore, the initial allocation is followed by some improvement heuristics which shuffles the jobs but gets benefit from families. Because the algorithm is a straightforward modification of the fair_share algorithm, we omit a more detailed description.

3.1.3 Random Allocation of Jobs

The allocation of jobs is random for each phase; the work load and the families are not considered. Random initial allocations are very fast to generate and they serve as initial solutions for the improvement methods.

a) Families in the LPTF order

b) Allocation of the families

Figure 4: Initial allocation of the families

3.2 Improvement of the Machine Allocation

In this section we give five swapping algorithms which can be used to improve the initial machine allocation determined by one of the three aforementioned techniques. A common characteristic of all these improvement methods is that we select always two machines in the same production phase and check whether we could improve the allocation by moving jobs from one machine to another. If this is possible, we move the job or jobs and repeat the procedure until a special termination criterion is met.

We must be careful when selecting the cost function by which we measure the effect of the moves. Because we aim at low set-up costs and a balanced work load; we search for an allocation for which the weighted sum of these

12

factors is minimal. A candidate allocation is realized if it has a low value of the *allocation cost* defined by:

$$\text{cost}_{lk} = c_w \cdot (|B_{nl}| \cdot t'_{nl} + |B_{nk}| \cdot t'_{nk}) + |W_{nl} - W_{nk}| \tag{2}$$

Here c_w is a parameter that controls the tendency of preserving the families. By increasing the value of c_w we increase the cost incurred from the set-up operations. The indices l and k refer to two machines of the phase n; $|B_{ni}|$ gives the number of different families of the machine i; t'_{ni} stands for the long set-up time and W_{ni} is the work load of the machine i ($i = l$ or k). The expression in the parentheses approximates long set-up times for the two machines, and the last term gives the imbalance of their work loads. W_{nl} is the largest and W_{nk} the smallest work load in a phase. The set of machines from which l and k are selected depends on the method.

Next we outline the five improvement algorithms for the allocation.

3.2.1 The Globally Best Pair Algorithm

Let us consider phase n ($1 \le n \le N$) and let l_0 and k_0 be the pair of machines for which the allocation cost (2) is maximal in this phase. The *Globally Best Pair* (Gbest) algorithm checks all machine pairs and tests whether a pairwise change of two jobs would decrease the current value of the maximal allocation cost ($\text{cost}_{l_0 k_0}$). If this is the case, we realize the change and start the same process again with an updated l_0, k_0 and $\text{cost}_{l_0 k_0}$. Note that the work loads in formula (2) stands for the maximal and minimal ones in the phase in question. Therefore, they are not necessarily the work loads of the current machine pair. Although we can perform the tests in an arbitrary order, we prefer an order where the index l runs from the machine with the maximal to the minimal work load. For each fixed l the index k runs in the opposite direction (i.e., from the minimal to the maximal work load), see Fig. 5.

The algorithm tests each pair of the jobs in the machines l and k. In addition to the job pairs, the algorithm tests the effect of moving a job from one machine to another (without returning to some other job), see Fig. 6.

The algorithm terminates after an unsuccessful round of iterations for all job pairs and machine pairs. We observe that the method Gbest is conservative in the sense that it realizes a move only if the globally defined maximal allocation cost (2) improves, see Fig. 7 and Fig. 8. Note that formula (2) includes not only work load imbalance but also another factor, namely the weighted set-up times. Therefore, a better value of (2) can be achieved even though the work load imbalance is not improved. Note that the illustrative
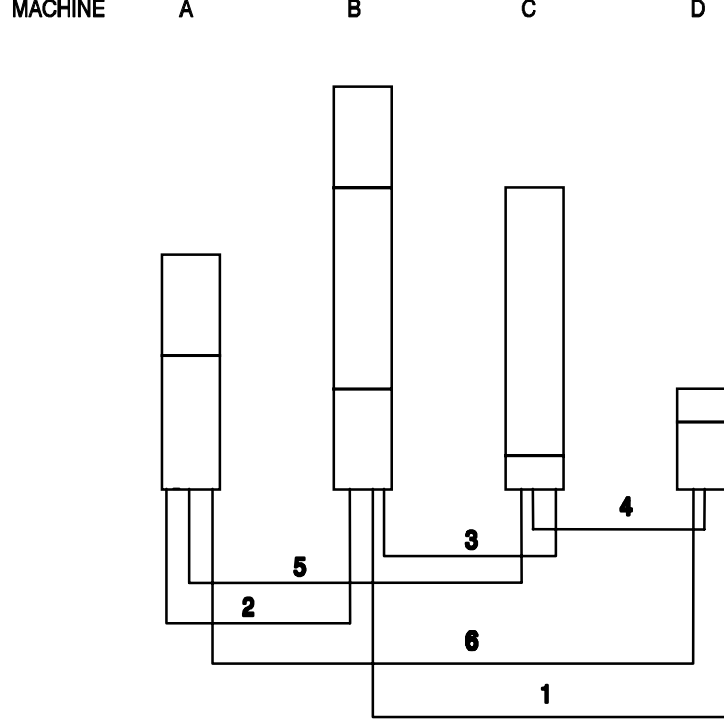
Figure 5: The order of the pairwise testing of the machines

examples of Fig. 7 and 8 omit the effect of the set-up times (that is the first factor in (2)) on the allocation cost.

Algorithm Gbest. The algorithm Gbest implements the idea of the Globally Best Pair changes. The method searches for the most profitable change in respect to (2). The swap is realized if it decreases the global allocation cost value. The selection-swapping phases are iterated until no improvement is achieved. The input of the algorithm is an initial allocation of the machines for each phase n $(n = 1, 2, \ldots, N)$:

G_{nm}	a set of jobs in the machine m
W_{nm}	work load of the machine m
B_{nm}	a set of families of the machine m
c_w	weighting coefficient of formula (2)
t'_{nm}	long set-up time

The algorithm determines an improved allocation given by updated G_{nm}, W_{nm}, B_{nm}.

Step 1 (Main loop) For all phases $n = 1, 2, \ldots, N$ perform the steps 2 to 6.
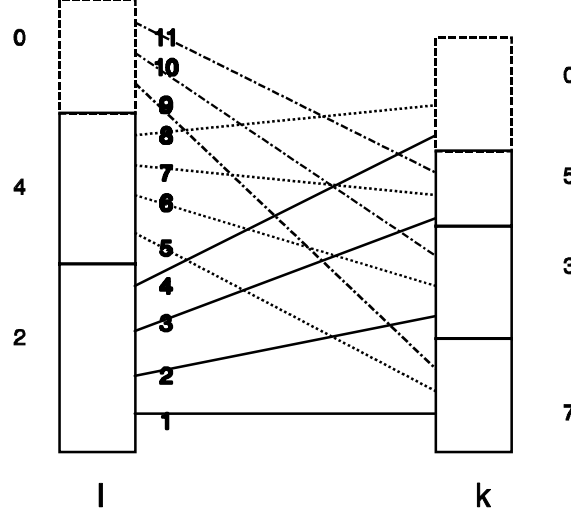
14

Figure 6: The testing of pairwise moves between two machines. Job 0 is nil; it is used to describe the move of one job only.

Step 2 (Initialization) Sort the machines m $(m = 1, 2, \ldots, M_n)$ into a decreasing order by the respective W_{nm} values.

Step 3 (Selection of the machine pair) While there are unconsidered machine pairs, select a new pair (l, k). We let l run through the indexes $1, 2, \ldots, M_{n-1}$ and for each l the index k runs through $M_n, M_{n-1}, \ldots,$ $l + 1$.

Step 4 (Test candidate moves) Determine the current allocation cost cost_{lk} of the machine pair l, k. (The work loads in formula (2) are calculated from all the machines in phase n.)

Step 5 (Improve allocation) For all pairs (i, j) of jobs in the machines l and k make a candidate move of job i from the machine l to the machine k and job j from the machine k to the machine l. Let $(G'_{nl}, W'_{nl}, B'_{nl})$ and $(G'_{nk}, W'_{nk}, B'_{nk})$ be the corresponding allocation of the two machines. Calculate $\mathsf{cost}'_{l,k}$ for the candidate move. If $\mathsf{cost}_{lk} > \mathsf{cost}'_{lk}$ then realize the move by updating the G, W, B settings and return to step 2 (i.e., repeat the testing of the machine pairs)

Step 6 (No improvement at steps 4 and 5) Terminate the algorithm.
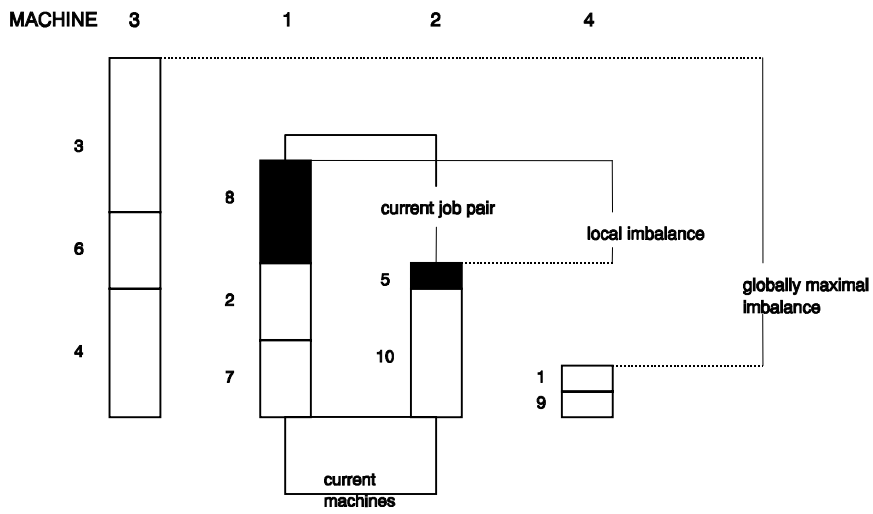
a) Before the move
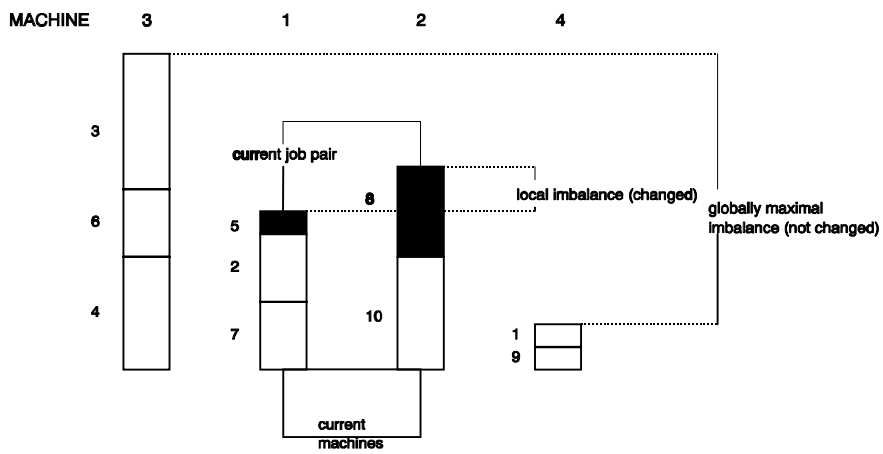
b) After the move

Figure 7: The effect of the swap of two jobs

a) Before the move

b) After the move

Figure 8: The use of the globally maximal allocation cost

17

3.2.2 Local Greedy Improvement

Algorithm Lgreedy. The idea of the *Local Greedy Method* (Lgreedy) is the same as in Gbest but the work loads in formula (2) are determined from the two machine in question and not from all the machines in phase n. A candidate move is realized if it gives a lower value of (2) than before the move, see Fig. 7 and 8 for the difference of the local and global approaches. The algorithm is a straightforward modification of Gbest.
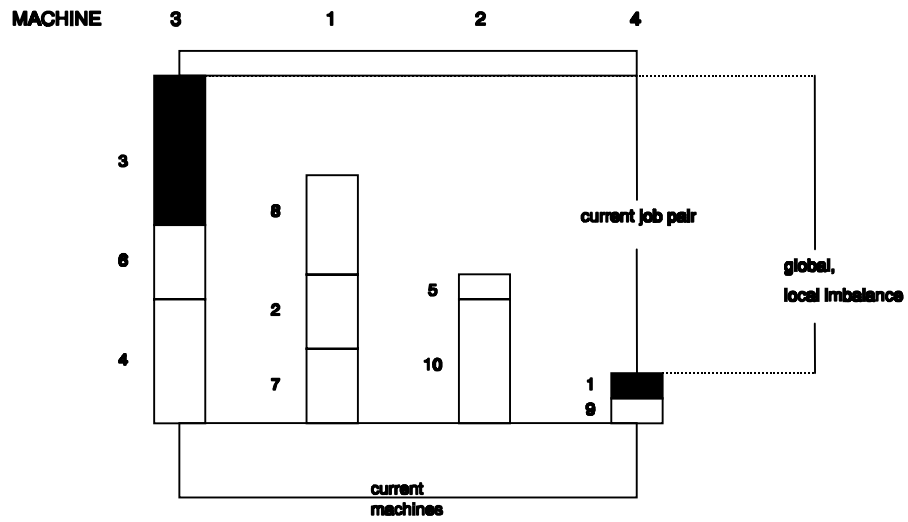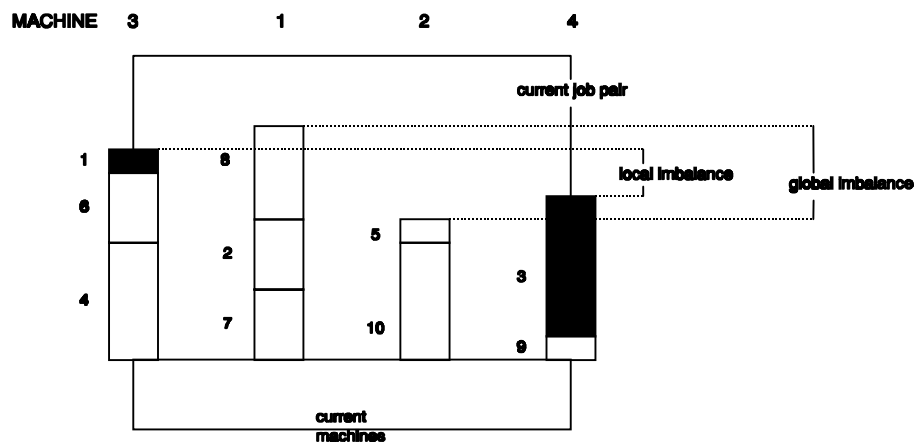
3.2.3 Pairwise Optimal, All Pairs

Algorithm Popt. There are important practical cases where the number of jobs allocated to each machine is rather small. In that case we can find the optimal pairwise allocations by applying enumeration. Let us assume that the initial allocation has placed m_l and m_k jobs to the machines l and k, respectively. Now we can perform the allocation of these $m = m_l + m_k$ jobs in 2^m different ways which is an exponentially increasing function of m. However, if m is less than a given number M_{limit}, we can calculate all possible allocations of the machine pair and select the best one. Again, we consider all possible machine pairs (l, k) but make a trade-off between the time and the efficiency because we consider now all the pairs only once. The problem of this method is that the comparison of the pair (i, l) can give a certain allocation which is then altered by the comparison of the machines (l, k). After that a repeated comparison of (i, l) could again change the allocations of i and l. Because of this we should further consider the convergence or calculate the overall cost of the allocation at each step. We omit this problem and concentrate on only one iteration for each pairwise comparison.

3.2.4 Pairwise Optimal, Random Pairs

Algorithm Ropt. This method differs from the above in the selection of the machine pairs only. We consider random machine pairs with replacement (i.e., we allow the reconsideration of the same pair). The number of the pairs is restricted by the user-defined parameter number_of_machine_pairs.

3.2.5 Selection by Function, Optimal Allocation

Algorithm Fopt. The idea of this algorithm is to consider the machine pairs in an order which increases our chances to make successful job moves. We calculate for each machine in phase n a *characteristic value*

$$r_m = c_{\text{family}} \cdot |B_{nm}| + c_{\text{job}} \cdot |G_{nm}| + c_{\text{work}} \cdot |W_{nm}| \qquad (3)$$

The user-defined coefficients c_{family}, c_{job} and c_{work} give the weighting to the number of the different families ($|B_{nm}|$), the number of the different jobs ($|G_{nm}|$) and the total work load ($|W_{nm}|$) of the machine m. The algorithm Fopt runs exactly like Ropt but now we choose at each selection step the machine pair (l, k) for which

$$\begin{cases} r_l = \min_{m \in \{1,\ldots,M_n\}} r_m, \\ r_k = \max_{m \in \{1,\ldots,M_n\}-\{l\}} r_m. \end{cases}$$

3.3   Initial Sequencing of the Jobs

Let us assume that the machines have been allocated for the jobs as described in section 3.1 and 3.2. The latter part of the scheduler sequences the jobs. This is done in two phases. In the first phase we determine an initial ordering for each machine separately, and in the second phase we try to improve this ordering by considering the other machines, too.

In this section we discuss one possible technique for the *initial sequencing*. We assume that the machine allocation (G_{nm}, W_{nm}, B_{nm}) has been given and recall that each job has its own data about the work load (w_{nj}), the family (f_{nj}) and the due date (d_j). Our task is to determine for each phase and machine pair (n, m) a sequence $\Pi(n, m) = (j_1, j_2, \ldots)$ which gives the processing order of the jobs.

The concept of families was introduced for minimizing the set-up times. Therefore, in the first step we group the jobs of the machine (n, m) according to the families and determine the sequence of the jobs within each family from the finishing times of the previous phase. In the second step we arrange the order of the families so that the idle time of the machine will be as short as possible.

Algorithm initial sequencing

Step 1 (Sequencing the jobs of a family): In the beginning the order of the jobs in a family is fixed to correspond to the order of the finishing times in the previous phase. The order of the previous phase should be preserved. However, it is possible that the finishing times of the previous phase are identical for two jobs of the family. (Note that the grouping into the families may be different in the different phases

and two different machines may process the job allocated to the same machine in the current phase. For the first phase the finishing times of the previous phase are all zero.) When the finishing times are equal, we can select the order by using one of the following rules (identified with a parameter setting):

1. Earliest due date first (EDF)
2. Longest job first (LJF)
3. Shortest job first (SJF)

If a rule fails we apply the remaining ones. If all rules fail, we select the job randomly.

Step 2 (Mutual ordering of the families): The mutual ordering of the families is based on the concept of *losses*. Let us denote the starting time of the job l in phase n by $\mathsf{starting\_time}_l(n)$ and its finishing time in the previous phase by $\mathsf{finishing\_time}_l(n-1)$. The starting time is the point where the processing of the job will start if the family of job l is selected as the next family. We define

$$\mathsf{loss}_l = \mathsf{finishing\_time}_l(n-1) - \mathsf{starting\_time}_l(n).$$

The loss gives us the amount of time during which a machine stays idle when the processing of the job is not completed in the previous phase. The loss of the whole family i is

$$\mathsf{loss}^{(i)} = \max_{f_{nl}=i}\{\mathsf{loss}_l, 0\}$$

(i.e., the maximum of the losses in the family i, or zero if all jobs are ready for processing in phase n). Note that when we calculate the starting times of the jobs we bear in mind that some families have already been scheduled and their processing loads have been added to the machine. The principal idea of the mutual sequencing is thus to *order the families into an increasing order of losses*.

Fig. 9 illustrates the calculation of losses when selecting the first family of a machine in phase n. The first job of each family has been placed on the time axis at its earliest point of starting. We select family 2 to be processed first

20
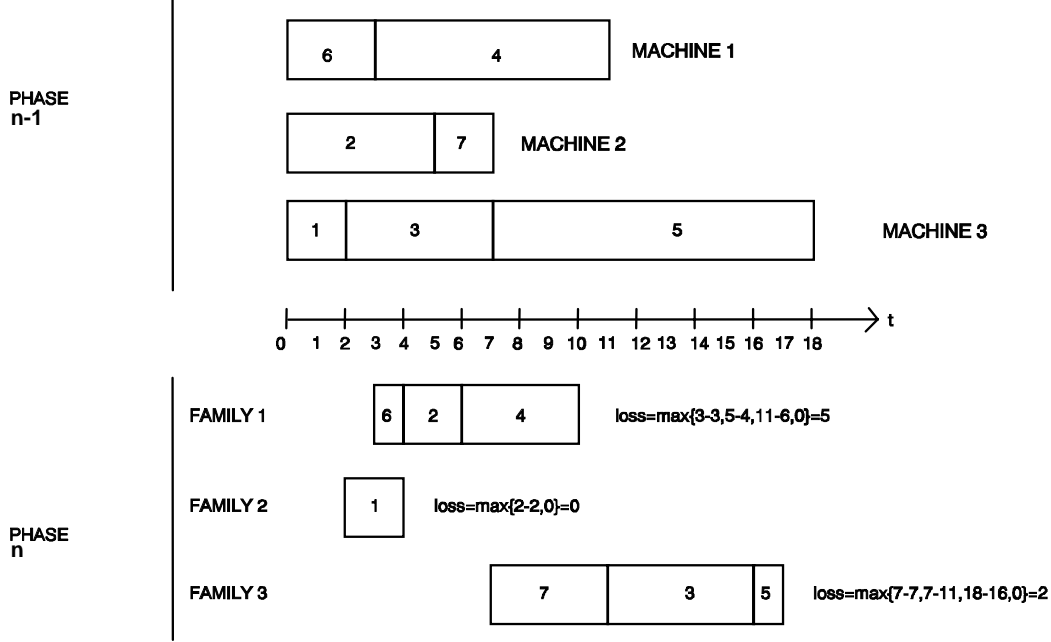
Figure 9: The use of losses in the scheduling of the families

and then we have to restart the consideration in the situation where there are two families left to be scheduled in the machine in question.

Again, it is possible that the use of the losses does not enforce an unambiguous ordering of the families. This is the case when several loss values are zero. Therefore, we introduce a selection criterion which is more global. Large urgent jobs are preferred. For this reason, we calculate for each family i the starting_number which is, for the *first phase, the average due date of the family weighted by the job sizes*, and for the *other phases the average finishing time of the previous work phases weighted by the job sizes*. The job size a_j is the number of PCBs in the job (or batch). Thus, for the family i we define

$$\text{starting\_number}_i = \begin{cases} \sum_j a_j \cdot \frac{d_j}{\sum a_j}, & \text{for the first phase,} \\ \sum_j a_j \cdot \frac{t_j}{\sum a_j}, & \text{for the other phases.} \end{cases} \tag{4}$$

The sum is over the jobs in the machine m in phase n, a_j is the job size, d_j is the due date and t_j the finishing time in the previous work phase. (The due date of the most urgent job is the zero level and other due dates are calculated in minutes from this. The jobs with no due date are given a due date that is one day after the last defined due date).

Fig. 10 illustrates the idea of a_j weighting in formula (4). The family 1 contains two urgent jobs and one small but not urgent job. Without weight-

21
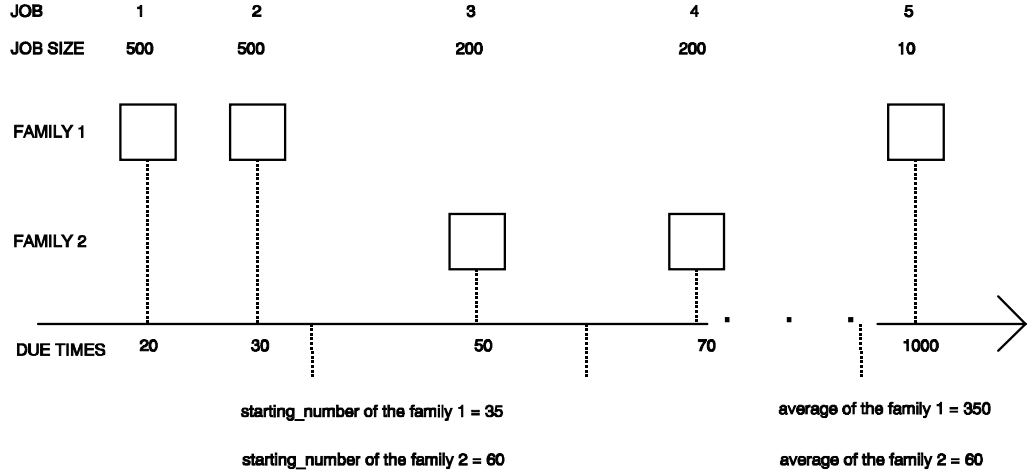
Figure 10: Calculation of the starting numbers

ing, family 2 would be scheduled before family 1. The method is heuristic and does not guarantee a correct ordering. Note that the finishing time in the previous work phase is the finishing time of a part of a batch consisting of a full transportation magazine (of, for example, $K = 30$ ready-made PCBs). Therefore, the machines of successive phases can process a given job in parallel:

$$\text{finishing\_time}_l(n-1) = \text{starting\_time}_l(n-1) + w_{n-1,l} \\ \cdot \min\{1, K/a_l\} + t_{nm,l-1,l}$$

where the last term stands for the set-up from the previous to the current job. The algorithm is quite straightforward and we omit its description.

3.4   The Improvement of a Schedule

A combination of the algorithms of the three previous subsections provides us with a machine allocation and an initial sequence of the jobs, denoted by G_{nm}, W_{nm}, B_{nm}, $\Pi(n,m)$ (allocation of the jobs, machine work load, families, ordering). In this section we introduce two more algorithms that can be used to improve the schedule. The principle is to make such changes to the schedule that the idle time of the machines will be reduced. There are two apparent methods for achieving this goal:

- re-scheduling whole families

- re-scheduling jobs in a family.

22
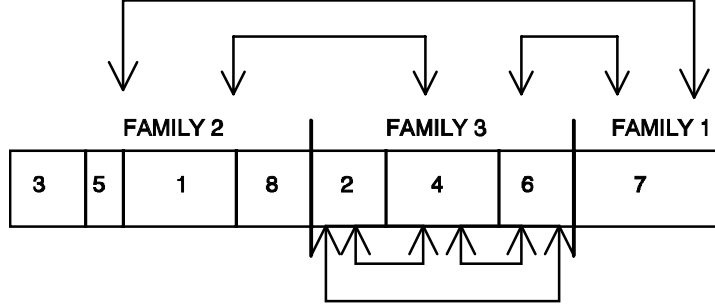
Figure 11: Pairwise comparisons of the families and the jobs

3.4.1 Re-scheduling Whole Families

Algorithm ReFamilies. The re-scheduling of families is done by considering all pairs of the families in a machine. The mutual ordering of jobs in a family is preserved, see the upper arrows of Fig. 11. The object function *machine waiting time*, mwt_{nm} counts the total elapsed idle time during the period from the starting of the first job until finishing of the last job in the machine. The idle time is used because some of the jobs from the previous phase are not ready when the current machine is ready to proceed. Note that we do not count the idle time before the first job and after the last job of the schedule because we assume that the operation of the production plant is continuous and these idle periods will be filled in by jobs of another production plan.

We perform the candidate move if it decreases the total machine idle time ($\sum_{n,m} \mathsf{mwt}_{nm}$) and does not violate the due dates. The same process is repeated after a move.

3.4.2 Re-scheduling of the Jobs in a Family

Algorithm ReJobs. The re-scheduling of the jobs in a family is performed as described above, see lower arrows in Fig. 11. The candidate moves do not break the integrity of the families.

Note that in ReFamilies and ReJobs the re-scheduling of the first line machine causes an automatic move of the jobs in the latter machine on the same line.

4 Practical Tests

In this section we apply the methods described in the previous section to scheduling problems arising from an actual production plant. To be more

23
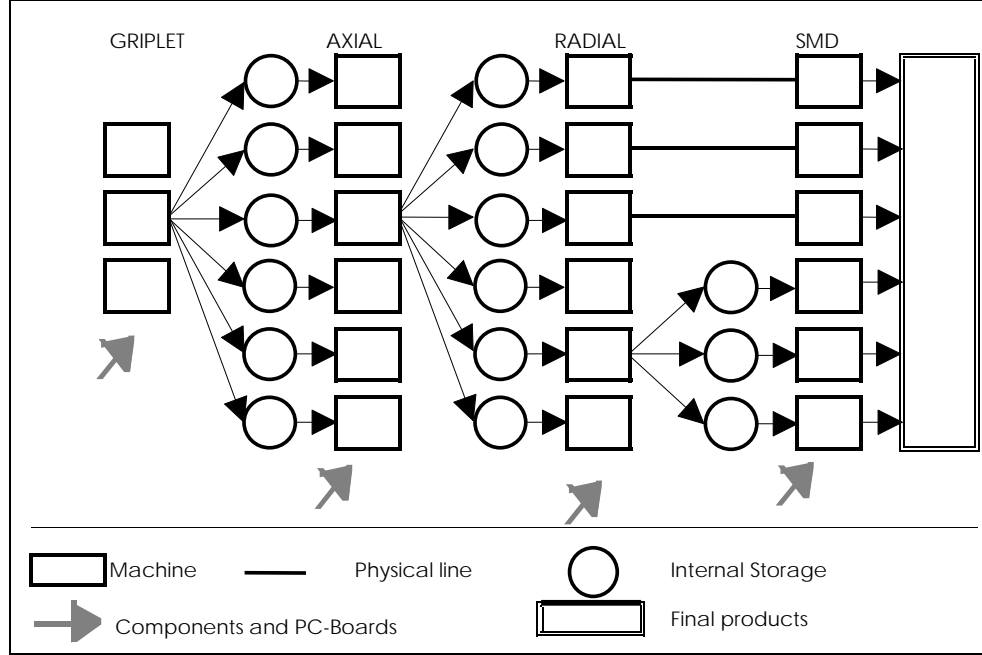
Figure 12: A sample machine configuration (Nokia Display Products, Salo)

specific, we consider a assembly plant for printing electronic components on *Printed Circuit Boards* (PCBs), where the production line consists of four successive phases:

1. Griplet insertion (GRIPLET)

2. Axial insertion (AXIAL)

3. Radial insertion (RADIAL)

4. Surface mounted onsertion (SMD)

In addition to these work phases there are preparative and subsequent operations that are not considered here. There are three fixed assembly lines and, moreover, it is possible to form logical lines. The machine configuration is shown in Fig. 12. The fixed lines are between RADIAL and SMD phases and a conveyor belt is used for transporting the PCBs, whereas the transportation is done in magazines between the other machines.

Table 1 gives an example of a typical production plan (PP) for one week (PPs usually comprise 20–35 different PCB types). In this case PP comprises 24 batches with varying due dates. The batch sizes range from 500 to 5,000. Furthermore, it is assumed that old batches are ready for processing

PCB	AMOUNT	DUE	PCB	AMOUNT	DUE
SH1275	800	12.01.	SH1299	2800	17.01.
SH1275	800	13.01.	SH1701	5000	15.01.
SH1275	800	14.01.	SH1708	800	12.01.
SH1275	800	15.01.	SH1714	2000	12.01.
SH1275	800	16.01.	SH1716	600	15.01.
SH1275	800	17.01.	SH1718	600	12.01.
SH1260	600	16.01.	SH1721	600	12.01.
SH1277	5000	15.01.	SH1722	500	15.01.
SH1294	2000	15.01.	SH1722	500	16.01.
SH1296	3600	12.01.	SH1722	500	17.01.
SH1298	900	12.01.	SH1703	4000	15.01.
SH1298	1000	16.01.	SH1262	1000	15.01.

Table 1: A sample production plan

at the beginning of the production period. The operation times for different machine-PCB-type combinations are calculated from the machine data and PCB data, or they are derived from the previous production data.

We consider the following features:

- the sum of the squared tardinesses,

- the sum of the internal storage levels (expressed as the number of PCBs),

- the sum of the internal waiting times (in minutes), and

- the number of PCB families constructed by the algorithm.

As mentioned earlier, the scheduling algorithm has been integrated with the IPS [8] by which the results can be visualized, see Fig. 13.

The graph for the internal storage level (Fig. 14) indicates that machine 16 (one of the SMD machines) is the bottleneck machine and more work load could be moved to machine 17 to balance this situation.

Four batches are late in this situation (Fig. 15). The worst situation is for batch 6 for which the violation of the due date is 24 hours. On the other hand, many of the jobs are bound to be finished much before their due dates. (Note that plje was not the best solution for minimizing tardiness.)

Table 2 shows some sample results of a schedule for the production plan in Table 1. We have studied some of the possible combinations of the scheduling algorithms. A variation in the results is observed also in the average storage

Figure 13: IPS main window. The system informs the production planner that batches SH1708, SH1275_2, SH1275_4, SH1722_1, SH1722_3, SH1275_1 will be late, more details can be viewed by clicking the corresponding graphical components.
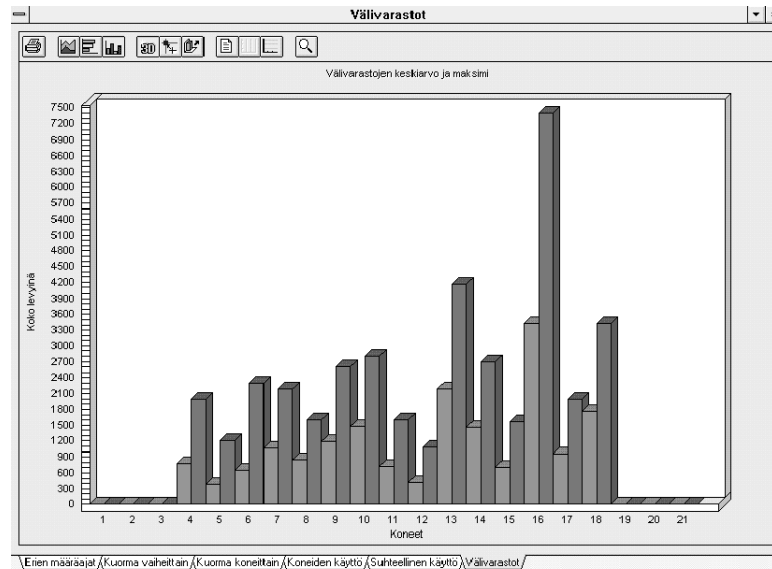
Figure 14: Internal storage graph for the solution plje. Dark piles stand for the maximal storage level and light piles for the average storage level. The indeces on the x-axis stand for the machines and y-axis gives the storage level given by the number of PCBs.

26
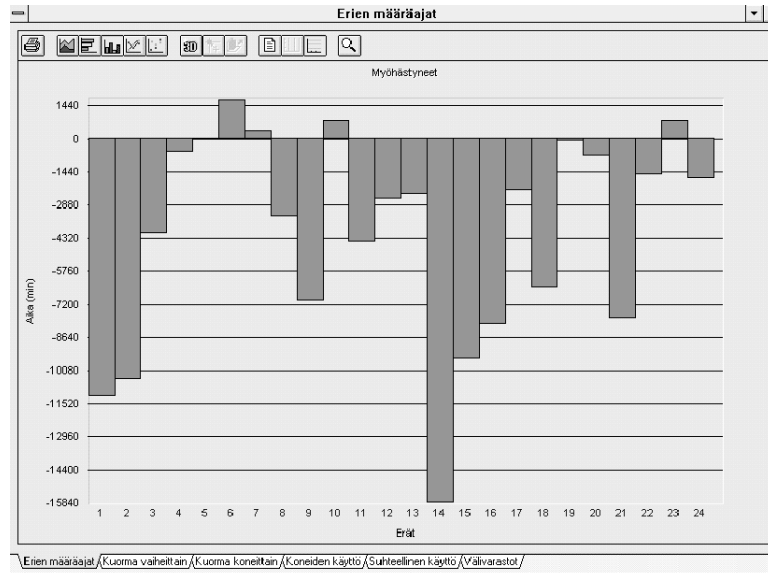
Figure 15: Lateness for the solution plje. The bars below the zero level represent jobs completed before the due dates whereas the jobs above it will be late. Indexing of the x-axis is for the PCB batches. The units of the y-axis is in minutes.
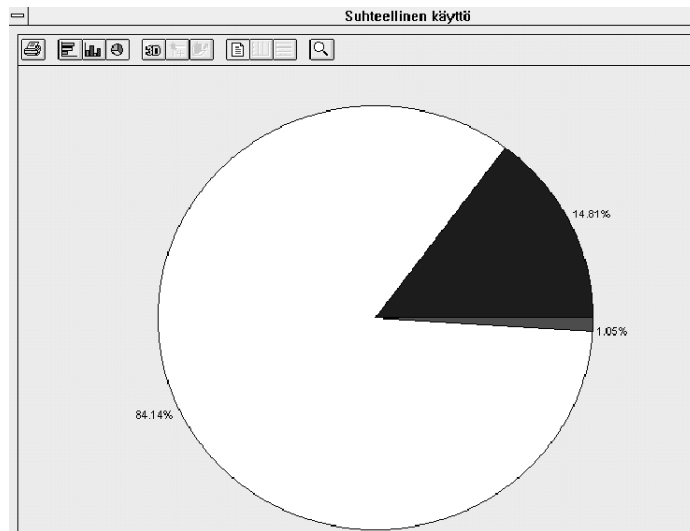
Figure 16: Machine usage for the solution plje. White area represents the ratio of effective processing, dark for the idle time and grey the set-up time.

27

used method	squared tardiness	average storage	waiting time	different families
akjf	156390	33040	34411	41
akje	156390	33040	34411	41
asjf	95625	36880	31136	37
asje	285317	32520	33541	37
avjf	248286	33420	33379	30
avje	254439	34920	31766	30
aljf	323081	29680	39157	41
alje	323081	29680	39157	41
agjf	114494	32600	29534	33
agje	138348	33720	26597	33
pljf	169655	38620	33035	42
plje	**169655**	**38620**	**33364**	**42**
pgjf	185578	42340	36952	39
pgje	197323	43800	30266	39
rkjf	87007	34920	30182	40
rkje	156261	36180	25951	41
rsjf	228784	37060	54460	36
rsje	220415	37320	35198	38
rvjf	134792	36160	29267	47
rvje	299283	29520	42272	39
rljf	144003	40460	26075	42
rlje	185554	35340	22435	40
rgjf	313361	33940	49364	39
rgje	227079	38880	27537	37

Table 2: Results for different algorithms for the sample problem of Table 1

level which ranges from 29,520 (rvje) to 43,800 (pgje). The sum of the waiting times ranges from 25,951 (pkje) to 54,460 (rsjf). The variation in the number of different families formed by the algorithm is relatively small (from 30 to 47). The squared tardiness of the schedules varies from 87,007 (rkjf) to 323,081 (aljf and alje). In addition to rkjf, the asjf technique finds a solution with a low tardiness value (95,625).

We used a second set of test runs to study the differences between the methods in respect to various features of the solution. The four test problem are described briefly in Table 3 (N.B., the table gives only production volumes and due dates and omits component types and production volumes). Three problems are based on actual production plans (with 24, 34 and 38 batches), and the fourth problem has been generated randomly from the production data to resemble a typical production plan. See figure 17 for a histogram of the methods.

Table 4 shows the ranking for the solution algorithms for the problem t04 when the problem is solved by 30 different variants of the algorithm (avjf,

Method	Squared tardiness				Internal storage				Idle times				Families				Total				
	t04	t08	t11	tsa	t04	t08	t11	tsa	t04	t08	t11	tsa	t04	t08	t11	tsa	t04	t08	t11	tsa	
agjf	30	30	27	10	12	20	26	6	17	20	28	6	25	22	25	16	30	30	27	10	97
aljf	29	28	26	12	8	21	19	8	16	9	14	8	16	18	4	18	29	28	26	12	95
rgjf	26	23	18	25	21	22	27	16	9	22	8	19	1	3	16	27	26	23	18	25	92
avjf	24	29	20	16	25	10	21	9	8	11	10	4	19	24	23	1	24	29	20	16	89
avje	23	24	19	22	24	7	20	23	12	23	9	15	18	23	22	2	23	24	19	22	88
agje	28	27	24	9	6	11	24	5	20	19	30	5	24	21	24	15	28	27	24	9	88
alje	27	25	25	11	4	15	18	7	21	21	13	7	15	17	3	17	27	25	25	11	88
asjf	19	26	13	28	16	24	8	25	10	24	27	2	7	20	9	4	19	26	13	28	86
rkje	17	19	14	30	7	6	17	3	23	29	17	18	6	10	10	5	17	19	14	30	80
rsjf	21	20	28	8	5	18	29	4	13	7	11	25	9	19	2	14	21	20	28	8	77
asje	16	22	9	29	1	19	2	26	18	30	25	26	17	16	17	9	16	22	9	29	73
akjf	15	21	23	14	19	12	7	15	27	13	20	9	12	8	19	19	15	21	23	14	73
rljf	8	7	30	27	18	8	25	24	24	28	7	3	4	4	13	28	8	7	30	27	72
rlje	18	15	16	18	10	14	13	18	15	25	22	22	3	2	12	23	18	15	16	18	67
rvjf	20	13	8	26	15	13	5	20	7	12	26	29	8	5	1	3	20	13	8	26	67
akje	13	18	21	15	20	5	4	22	28	14	21	24	10	7	18	21	13	18	21	15	67
rkjf	22	17	22	2	28	16	12	21	11	10	18	1	5	9	8	7	22	17	22	2	63
rvje	25	16	15	6	17	23	11	1	22	27	19	30	20	6	11	8	25	16	15	6	62
rgje	9	14	17	17	14	25	14	17	30	26	12	21	2	1	20	22	9	14	17	17	57
rsje	7	8	29	1	2	9	22	2	29	8	29	16	11	11	5	6	7	8	29	1	45
pljf	14	10	12	7	13	2	10	13	14	15	24	12	14	12	7	13	14	10	12	7	43
pkjf	6	6	3	24	30	29	16	30	2	6	5	14	30	28	30	30	6	6	3	24	39
pgjf	11	12	10	5	9	3	6	12	25	17	15	28	22	14	15	12	11	12	10	5	38
pkje	5	5	2	23	29	30	15	29	3	5	4	13	29	27	29	29	5	5	2	23	35
plje	12	9	11	3	3	1	9	10	19	16	23	20	13	13	6	10	12	9	11	3	35
pgje	10	11	7	4	11	4	3	11	26	18	16	27	23	15	14	11	10	11	7	4	32
pvjf	2	2	5	21	22	26	30	28	5	3	1	11	26	29	28	26	2	2	5	21	30
psje	4	4	1	19	26	28	1	19	1	2	2	23	28	26	21	24	4	4	1	19	28
pvje	1	1	4	20	23	27	28	27	6	4	3	10	27	30	27	25	1	1	4	20	26
psjf	3	3	6	13	27	17	23	14	4	1	6	17	21	25	26	20	3	3	6	13	25

a) Weights of (squared tardiness, internal storage level, internal waiting times, number of families) are $(1, 0, 0, 0)$; i.e., the last column equals to the rank for squared tardiness. Rank 30 stands for the best method, 1 for the worst.

Method	Squared tardiness				Internal storage				Idle times				Families				Total				
	t04	t08	t11	tsa	t04	t08	t11	tsa	t04	t08	t11	tsa	t04	t08	t11	tsa	t04	t08	t11	tsa	
rgjf	26	23	18	25	21	22	27	16	9	22	8	19	1	3	16	27	47	45	45	41	178
avje	23	24	19	22	24	7	20	23	12	23	9	15	18	23	22	2	47	31	39	45	162
agjf	30	30	27	10	12	20	26	6	17	20	28	6	25	22	25	16	42	50	53	16	161
asjf	19	26	13	28	16	24	8	25	10	24	27	2	7	20	9	4	35	50	21	53	159
avjf	24	29	20	16	25	10	21	9	8	11	10	4	19	24	23	1	49	39	41	25	154
aljf	29	28	26	12	8	21	19	8	16	9	14	8	16	18	4	18	37	49	45	20	151
rljf	8	7	30	27	18	8	25	24	24	28	7	3	4	4	13	28	26	15	55	51	147
pkjf	6	6	3	24	30	29	16	30	2	6	5	14	30	28	30	30	36	35	19	54	144
rkjf	22	17	22	2	28	16	12	21	11	10	18	1	5	9	8	7	50	33	34	23	140
pkje	5	5	2	23	29	30	15	29	3	5	4	13	29	27	29	29	34	35	17	52	138
pvjf	2	2	5	21	22	26	30	28	5	3	1	11	26	29	28	26	24	28	35	49	136
agje	28	27	24	9	6	11	24	5	20	19	30	5	24	21	24	15	34	38	48	14	134
rsjf	21	20	28	8	5	18	29	4	13	7	11	25	9	19	2	14	26	38	57	12	133
alje	27	25	25	11	4	15	18	7	21	21	13	7	15	17	3	17	31	40	43	18	132
pvje	1	1	4	20	23	27	28	27	6	4	3	10	27	30	27	25	24	28	32	47	131
rgje	9	14	17	17	14	25	14	17	30	26	12	21	2	1	20	22	23	39	31	34	127
akjf	15	21	23	14	19	12	7	15	27	13	20	9	12	8	19	19	34	33	30	29	126
asje	16	22	9	29	1	19	2	26	18	30	25	26	17	16	17	9	17	41	11	55	124
rlje	18	15	16	18	10	14	13	18	15	25	22	22	3	2	12	23	28	29	29	36	122
rvjf	20	13	8	26	15	13	5	20	7	12	26	29	8	5	1	3	35	26	13	46	120
akje	13	18	21	15	20	5	4	22	28	14	21	24	10	7	18	21	33	23	25	37	118
rvje	25	16	15	6	17	23	11	1	22	27	19	30	20	6	11	8	42	39	26	7	114
rkje	17	19	14	30	7	6	17	3	23	29	17	18	6	10	10	5	24	25	31	33	113
psjf	3	3	6	13	27	17	23	14	4	1	6	17	21	25	26	20	30	20	29	27	106
psje	4	4	1	19	26	28	1	19	1	2	2	23	28	26	21	24	30	32	2	38	102
pljf	14	10	12	7	13	2	10	13	14	15	24	12	14	12	7	13	27	12	22	20	81
rsje	7	8	29	1	2	9	22	2	29	8	29	16	11	11	5	6	9	17	51	3	80
pgjf	11	12	10	5	9	3	6	12	25	17	15	28	22	14	15	12	20	15	16	17	68
pgje	10	11	7	4	11	4	3	11	26	18	16	27	23	15	14	11	21	15	10	15	61
plje	12	9	11	3	3	1	9	10	19	16	23	20	13	13	6	10	15	10	20	13	58

b) Weights are $(1, 1, 0, 0)$

Table 3: Ranking of the solution algorithms for the test problems t04, t08, t11 and tsa (random)
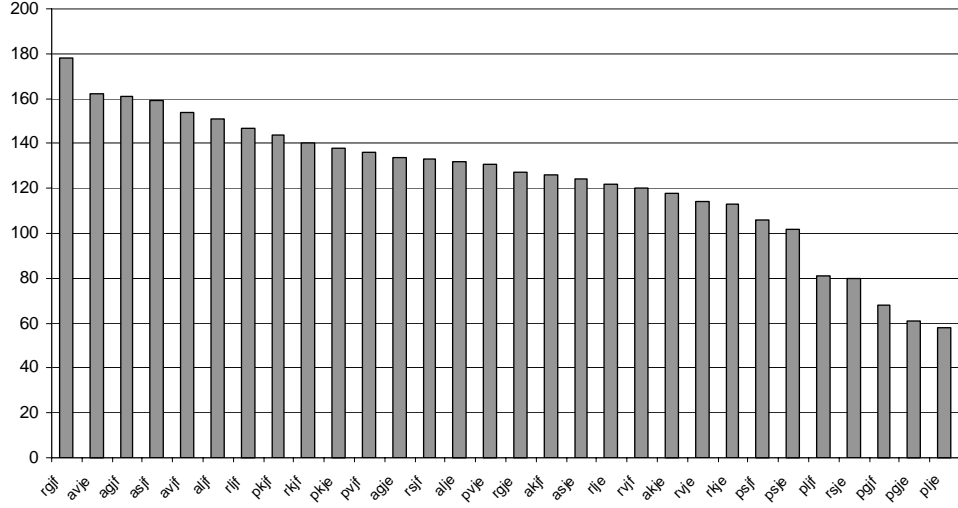
Figure 17: Histogram of method rankings

rkjf, ...). We rank each method in such a way that 30 is assigned to the best solution algorithm, 29 to the second best and so forth. Each feature is ranked individually, and the sum of ranks (in the last column) characterizes the optimality of the algorithm. The ranks can associated with different weights in order differentiate the importance of the features. If we consider only squared tardinesses (Table 4a) then the agjf variant appears to be the best for problem t04. Moreover, this solution has works relatively well also for the other features (internal storage, internal waiting times and subfamilies are ranked 12, 17 and 25, respectively). The results when squared tardinesses and internal storage are weighted equally are show in Table 4b.

Instead of concentrating only on one scheduling problem, we can easily make a summary of the rankings for a set of test problems. Table 3 includes the results for the four test problems and the last columns gives the weighted sum of all the ranks in the row. The overall best squared tardiness is now for agjf (see Table 3a). If we give equal weights to the squared tardiness and the internal storage level the best method is rgjf (see Table 3b). In this case the weakest score is for plje that has a rank below 15 in 16 columns out of 20.

Method	Squared tardiness	Internal Storage	Idle times	Families	Total
agjf	30	12	17	25	30
aljf	29	8	16	16	29
agje	28	6	20	24	28
alje	27	4	21	15	27
rgjf	26	21	9	1	26
rvje	25	17	22	20	25
avjf	24	25	8	19	24
avje	23	24	12	18	23
rkjf	22	28	11	5	22
rsjf	21	5	13	9	21
rvjf	20	15	7	8	20
asjf	19	16	10	7	19
rlje	18	10	15	3	18
rkje	17	7	23	6	17
asje	16	1	18	17	16
akjf	15	19	27	12	15
pljf	14	13	14	14	14
akje	13	20	28	10	13
plje	12	3	19	13	12
pgjf	11	9	25	22	11
pgje	10	11	26	23	10
rgje	9	14	30	2	9
rljf	8	18	24	4	8
rsje	7	2	29	11	7
pkjf	6	30	2	30	6
pkje	5	29	3	29	5
psje	4	26	1	28	4
psjf	3	27	4	21	3
pvjf	2	22	5	26	2
pvje	1	23	6	27	1

a) Weights $(1, 0, 0, 0)$

Method	Squared tardiness	Internal Storage	Idle times	Families	Total
rkjf	22	28	11	5	50
avjf	24	25	8	19	49
rgjf	26	21	9	1	47
avje	23	24	12	18	47
agjf	30	12	17	25	42
rvje	25	17	22	20	42
aljf	29	8	16	16	37
pkjf	6	30	2	30	36
rvjf	20	15	7	8	35
asjf	19	16	10	7	35
pkje	5	29	3	29	34
agje	28	6	20	24	34
akjf	15	19	27	12	34
akje	13	20	28	10	33
alje	27	4	21	15	31
psje	4	26	1	28	30
psjf	3	27	4	21	30
rlje	18	10	15	3	28
pljf	14	13	14	14	27
rsjf	21	5	13	9	26
rljf	8	18	24	4	26
pvjf	2	22	5	26	24
pvje	1	23	6	27	24
rkje	17	7	23	6	24
rgje	9	14	30	2	23
pgje	10	11	26	23	21
pgjf	11	9	25	22	20
asje	16	1	18	17	17
plje	12	3	19	13	15
rsje	7	2	29	11	9

b) Weights $(1, 1, 0, 0)$

Table 4: Ranking of the solution algorithms for the problem t04

5 Concluding Remarks

Most papers on flow shop scheduling have expressed the problem solving as an oversimplified procedure where the solution algorithm manages the whole scheduling situation without any interaction by the production engineer. Our approach is more realistic and interactive. Because the objectives of the scheduling are conflicting, we have included in our algorithm the possibility to stress different aspects freely. The scheduler includes a number of different methods which aim at somewhat different goals. In addition, the importance of human integration is emphasized by the introduction of IPS which gives the production designer a full control of the manufacturing process.

One difficulty in the development and use of the algorithms is the selection of the weighting parameters. The designer is encouraged to experiment with different settings of the parameters but we have assigned a default value to each of them. In most cases the selection of these values is rather easy, see Appendix A. If due dates are considered important, then a setting which favors this objective can be selected, or one can avoid long setting times by stressing the family concept. The next step in the development of our algorithm will be the inclusion of rules of fuzzy logic and self-adaptation to the optimizer.

Acknowledgment

References

[1] AMMONS, J. C., CARLYLE, M., CRANMER, L., DEPUY, G., ELLIS, K., McGINNIS, L. F., TOVEY, C. A., AND XU, H. Component allocation to balance workload in printed circuit card assembly systems. *IIE Transactions 29*, 4 (1997), 265–75.

[2] ASKIN, R. G., DROR, M., AND VAKHARIA, A. J. Printed circuit board family grouping and component allocation for a multimachine, open-shop assembly cell. *Naval Research Logistics 41* (1994), 587–608.

[3] BRUCKER, P. *Scheduling Algorithms*. Springer-Verlag, 1995.

[4] CRAMA, Y., FLIPPO, O. E., VAN DE KLUNDERT, J., AND SPIEKSMA, F. C. R. The assembly of printed circuit boards: A case with multiple machines and multiple board types. *European Journal of Operational Research 98*, 3 (1997), 457–72.

[5] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, 1979.

[6] HÄYRINEN, T. Subsequent workphases and parallel machines — scheduling algorithms for efficient control of PC-board assembly system. Master's thesis, University of Turku, 1996. (In Finnish: Peräkkäiset työvaiheet ja rinnakkaiset koneet — töidenjärjestelyalgoritmeja piirilevyjen koneladonnan tehostamiseksi).

[7] JOHNSSON, M., PELTONEN, S., LEIPÄLÄ, T., AND NEVALAINEN, O. Work load balancing of a generalized flexible flow line in printed circuit board production. In *Proc Fifth IASTED Int Conf, Robotics and Manufacturing* (Cancun, Mexico, May 1997), R. V. Mayorga, Ed., IASTED, IASTED/ACTA PRESS, pp. 382–9.

[8] JOHTELA, T., SMED, J., JOHNSSON, M., LEHTINEN, R., AND NEVALAINEN, O. Supporting production planning by production process simulation. *Computer Integrated Manufacturing Systems 10*, 3 (1997), 193–203.

[9] KIM, Y.-D., LIM, H.-G., AND PARK, M.-W. Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. *European Journal of Operational Research 91* (1996), 124–43.

[10] KUMAR, K. R., KUSIAK, A., AND VANELLI, A. Grouping of parts and components in flexible manufacturing systems. *European Journal of Operational Research 24* (1986), 387–97.

[11] KUSIAK, A. Application of operational research models and techniques in flexible manufacturing systems. *European Journal of Operational Research 24* (1986), 336–45.

[12] SHEVELL, S. F., BUZACOTT, J. A., AND MAGAZINE, M. J. Simulation and analysis of a circuit board manufacturing facility. In *Procedings of the 1986 Winter Simulation Conference* (1986), J. Wilson, J. Henriksen, and S. Roberts, Eds., pp. 686–93.

[13] SMED, J., JOHNSSON, M., PURANEN, M., LEIPÄLÄ, T., AND NEVALAINEN, O. Job grouping in surface mounted component printing. Tech. Rep. 196, Turku Centre for Computer Science, Aug. 1998. (Submitted to publication).

[14] STECKE, K. E. A hierarchical approach to solving machine grouping and loading problems of flexible manufacturing systems. *European Journal of Operational Research 24* (1986), 369–78.

[15] WITTROCK, R. J. An adaptable scheduling algorithm for flexible flow lines. *Operations Research 36*, 3 (1988), 445–53.

A Summary of Notations

n	phase
N	number of phases
m	machine
j	job, batch, product
J	number of jobs
M_n	number of machines of phase n
w_{nj}	the amount of work to be performed in phase n to product j
t_{nmjk}	the set-up time from job j to job k on machine m of phase n
f_{nj}	the family of job j in phase n
d_j	the due date of job j
F_{ni}	the set of jobs belonging to family i of phase n
H_{nml}	1 if the machine m of phase n and the machine l of the phase $n+1$ are connected by a fixed conveyor belt; 0 otherwise
B_n	the set of families of phase n
B_{nml}	the set of families of machine m of phase n
W_{nm}	the work load of machine m of phase n
c_0	fair share coefficient of formula (1)
a_j	the size of job (batch) j
K	the size of a magazine
G_{nm}	the set of jobs allocated to machine m of phase n
c_w	coefficient for family in formula (2)
cost_{lk}	allocation cost (2) of two machines l and k
fair_share_n	fair share parameter (1) of phase n
share_excess	share excess parameter
loss_j	the loss value of job j
M_{limit}	parameter for the number of jobs
$\text{number\_of\_machine\_pairs}$	constant for the number of machine pairs
c_{family}	constant for family in formula (3)
c_{job}	constant for job in formula (3)
c_{work}	constant for work in formula (3)
$\text{finishing\_time}_l(n)$	the finishing time of job l in phase n
$\text{starting\_time}_l(n)$	the starting time of job l in phase n
starting_number	criterion (4) for sequencing the families

$\Pi(n, m)$	the schedule of the machine m of phase n	
mwt$_{nm}$	machine waiting time of machine m of phase n	
r_m	criterion (3) for selecting machines	

Default values for parameters

c_0	fair share coefficient of formula (1)	20
K	the size of magazine	20
c_w	coefficient for family in formula (2)	3
share_excess	share excess parameter	50
M_{limit}	parameter for the number of jobs	15
number_of_machine_pairs	constant for the number of machine pairs	50
c_{family}	constant for family in formula (3)	1
c_{job}	constant for job in formula (3)	1
c_{work}	constant for work in formula (3)	1