Minimizing weighted total earliness, total tardiness and setup costs

H.A. ten Kate[1]
J. Wijngaard[2]
W.H.M. Zijm[3]

SOM theme A : Structure, Control, and Organization of Primary Processes

Abstract

A single machine scheduling problem is described which covers earliness costs, tardiness costs and setup costs. Setups are needed whenever a switch is made to a job of another product family. All jobs within the same product family have the same processing time. Besides setup costs also setup times are involved. Idle time is allowed, preemption is not allowed. Both a mixed integer programming formulation and a Branch and Bound approach are presented which solve the problem optimally. Experimental results are given for a number of heuristic methods.

---

[1] Faculty of Management and Organization, University of Groningen, P.O. box 800, 9700 AV, The Netherlands. This research was sponsored by the (former) Economics Research Foundation, which is part of the Netherlands Organization for Scientific Research (NWO).

[2] Faculty of Management and Organization, University of Groningen, The Netherlands.

[3] Department of Mechanical Engineering, University of Twente, P.O. Box 217, 7500 AE, Enschede, The Netherlands.

1. Introduction

Consider a production-to-order system in which the product range can be decomposed to a number of product families. Between production of orders for products belonging to the same family almost no setup is required, whereas a serious setup is incurred between orders for products belonging to different families. Hence, for reasons of efficiency, we prefer to continue with orders for products belonging to the same family as long as possible. However, the need to finish orders as close as possible to their required due dates (as is the ultimate goal in Just In Time manufacturing systems) may conflict with the efficiency objective. In general, a trade-off has to be made between efficiency on the one hand and a high degree of customer service on the other hand.

In a dynamic environment, the set of orders to be produced is changing every time a new order arrives or a scheduled order is completed. In such an environment one often works according to a "rolling horizon" method, where a detailed schedule is presented for a fixed set of orders which is maintained for a certain period of time, after which the schedule is updated. Questions about the length of the horizon and the length of the review period then naturally arise, but the basic underlying scheduling problem is a static one: scheduling a fixed set of orders such that due-dates are met as close as possible, taking into account the family structure for reasons of efficiency.

In Ten Kate[1994] a simulation model is described which is used to analyze the order acceptance in such production systems. Order acceptance is required to control the total set of orders which have to be scheduled. This is necessary because otherwise too many orders may be accepted for a short period of time, and even the best solution for the static scheduling problem will result in a bad performance for the dynamic problem. The simulation model has been simplified as far as possible in order to facilitate the analysis. Still, the main characteristics of the model are a good representation of the essential factors of the above described production systems.

In the simulation model as well as in practice fast but good heuristic methods are needed to solve the static scheduling problem. This paper is devoted

to finding such heuristic methods for the scheduling problem encountered in the simulation model. It is a first step towards finding such heuristics for the general static scheduling problem in these kind of production systems.

We consider a single machine production system in which production is to order. Jobs are characterized by their type (the family they belong to) and their due date. We assume that jobs of the same type have the same processing time and setup time. Each time we start producing jobs of a type different from the one just completed, a setup is required. Jobs finished too early must be kept in stock until their due date. Earliness costs have to be paid for each time unit such a job is early. Jobs finished too late lead to backlogging and customer dissatisfaction. Therefore, for each time unit a job is finished too late, tardiness costs are used to represent the costs of backlogging and customer dissatisfaction. Tardiness costs are assumed to be higher than earliness costs. For reasons of efficiency we prefer to have a small number of setups, and we assume that setup costs have to be paid every time a setup is needed. Our objective is to minimize the total costs. We assume that idle time is allowed and that preemption is forbidden.

Note that setup costs are equal across all job types, tardiness costs are equal across all job types, and earliness costs are equal across all job types. In any dynamic environment there is a classical trade-off between efficiency on the one hand and a timing aspect on the other hand. Due to the efficiency aspect jobs should be clustered over a longer period of time, whereas the timing aspect desires to produce the jobs as close as possible to the desired due dates, in order to meet due dates and at the same time keeping the inventories low. The cost parameters used in the simulation model are merely used as intermediate variables for both aspects in the above mentioned trade-off and are not suggested to reflect real-life costs. The setup cost parameter is used to reflect the need for efficiency, whereas the earliness and tardiness cost parameters are used for the timing aspect. Choosing them independent from jobs or job types is sufficient for controlling the above mentioned trade-off.

The sequel of this paper is organized as follows. In section 2 an overview of related problems in the literature is presented. In section 3 general properties of the static scheduling are discussed and it is described how this problem may be

solved optimally. Since solving the problem optimally can take a long computation time if the set of jobs to be scheduled is large, in the simulations a heuristic scheduling procedure has to be used. In section 4 various heuristic scheduling procedures are described. They are tested in section 5, where also the (hybrid) scheduling procedure is described which is used in the simulations.

2. Literature overview

Scheduling problems which contain some of the characteristics described above are scattered throughout the literature. Gupta and Kyparisis[1987] review a number of important ones. Baker and Scudder[1990] present a review on scheduling problems with earliness and tardiness penalties. They discern two major classes of problems. The first class involves a common due date for all jobs. This problem is in the second class, which permits the due dates to differ. Ow and Morton[1989] examine the problem of minimizing total earliness and tardiness costs and present some interesting heuristic methods. However, they do not allow idle time, which is inconsistent with the earliness/tardiness criterion (Baker and Scudder[1990]). Garey et al.[1988] also consider the problem of minimizing the total earliness and tardiness, and they do allow idle time. For problems with all tasks having the same length as well as for problems with a fixed sequence they discuss efficient algorithms to find an optimal solution.

As is recognized in Baker and Scudder[1990], the search for optimal schedules may be decomposed in two sub-problems : finding a good job sequence and scheduling inserted idle time. Fry et al.[1987] consider the problem of minimizing weighted total earliness and tardiness. They present a method to optimally insert idle time, and give empirical results for some heuristics. Yano and Kim[1991] consider the problem of minimizing weighted earliness and tardiness. They present a dynamic programming approach for inserting idle time. Furthermore, they compare the solutions of a number of heuristic methods with the optimal solutions, which are found by a Branch and Bound procedure. For the earliness/tardiness problem with general sequence-dependent setup times Coleman[1992] describes a mixed integer programming formulation.

Some papers address batch setup times: Zdrzalka[1991] considers a problem with unit batch setup times and delivery times. The objective is to minimize the maximum delivery time, which is equivalent with minimizing maximum lateness. Monma and Potts[1989] consider the complexity of a number of scheduling problems with batch setup times. For single machine problems they consider maximum lateness, total weighted completion times and number of late jobs criteria. The scheduling problem with batch setup times is related to the lot-sizing problem. Potts and Van Wassenhove[1992] present a review on the integration of both. When holding costs are assumed, the problem comes close to the problem as described here. Woodruff and Spearman[1992] describe a tabu search method which basically concerns this problem. However, instead of allowing jobs to be tardy, they use the due dates as deadlines. The same holds for Jordan and Drexl[1994] who consider a comparable problem. Since the problem can be interpreted as a scheduling problem as well as a lot-sizing problem, they compare solution methods for both approaches. They show that the solution methods which interpret the problem as a scheduling problem are faster than the solution methods which interpret the problem as lot-sizing problem.

The problem in this text combines earliness costs and tardiness costs with batch setup times and, moreover, setup costs. As far as is known, no attention has been given to this particular problem so far.

3. The problem

3.1 Formal description

The static scheduling problem can be described as follows : Given a set S, containing m jobs which belong to F product families ($F \leq m$; S contains m_f jobs of family f). Each job J_{fj} ($j \in \{1..m_f\}$, $f \in \{1..F\}$) is characterized by its family f and its due date d_{fj}. For each family f ($f \in \{1..F\}$) a processing time p_f and a setup time s_f are given. A setup is required every time we start to produce another family, whereas no setup is needed between two products of the same family.

By any schedule $\sigma(S)$ for S a completion time $C_{fj,\sigma}$ is determined for every J_{fj}. From $C_{fj,\sigma}$ earliness is defined as $E_{fj,\sigma} = \max\{0, d_{fj} - C_{fj,\sigma}\}$, and tardiness is defined

as $T_{fj,\sigma} = \max\{0, C_{fj,\sigma} - d_{fj}\}$. In the remainder of this text the index σ will be skipped for all schedule-related variables (E_{fj}, T_{fj}, C_{fj} and other), unless ambiguity could arise. The objective is to find a schedule $\sigma(S)$ in which the total weighted costs, consisting of earliness, tardiness and setup costs, are minimal. Thereby the earliness costs per order per unit of time are denoted by α, the tardiness costs per order per unit of time are denoted by β, and the costs per setup are denoted by γ. A binary variable τ_{fj} is used which is equal to 1 if a setup is needed before the production of job J_{fj}, and which is equal to 0 otherwise. The total weighted costs thus are :

$$\sum_{f=1}^{F} \sum_{j=1}^{m_f} \left[\alpha * E_{fj} + \beta * T_{fj} + \gamma * \tau_{fj} \right]$$

Idleness of the machine between the production of two jobs is permitted but preemption of the production of a job is not, i.e. once we have started producing a job, we must complete it without any further interruption.

3.2 Inserting idle time

Since this problem is a generalization of the problem considered in Garey et al.[1988] (take F=m, s_f=0, α=β=1, γ=0), we know that the problem is NP-hard. However, given a fixed sequence we can optimally insert idle time: for a fixed sequence setup costs are fixed, and for all jobs the processing time plus the possible setup time is fixed. Therefore, for a fixed sequence the problem is equivalent to the problem considered in Fry et al.[1987]. The linear programming formulation below gives an optimal solution for inserting idle time in a fixed sequence of jobs. It is due to Fry et al.[1987], but has been reformulated in terms of the notation used in this text. Remark that instead of job-dependent earliness and tardiness costs, as in Fry et al.[1987], equal penalties for all jobs are used here.

In this formulation the following notation is used : Consider a sequence π of all jobs in S. Let $\pi[i]$ denote the i^{th} job in the sequence π. Denote by q_{fj} the sum of processing time and, if necessary, setup time. For all job-related variables it holds that if $\pi[i]=J_{fj}$ then, for example, E_{fj} can also be denoted by $E_{\pi[i]}$. From any sequence $\pi(S)$ the schedule $\sigma(S)$ is obtained by optimally inserting idle time.

Linear programming to insert idle time (LPIT)

$$\text{Minimize} \quad \sum_{i=1}^{m} \alpha E_{\pi[i]} + \beta T_{\pi[i]} \qquad \text{(LPIT 0)}$$

subject to

$$C_{\pi[i]} = C_{\pi[i-1]} + q_{\pi[i]} + I_{\pi[i]} \qquad \forall\, i \in \{1..m\} \qquad \text{(LPIT 1)}$$

$$T_{\pi[i]} - E_{\pi[i]} = C_{\pi[i]} - d_{\pi[i]} \qquad \forall\, i \in \{1..m\} \qquad \text{(LPIT 2)}$$

$$C_{\pi[0]} := 0 \qquad \text{(LPIT 3)}$$

$$E_{\pi[i]},\ T_{\pi[i]},\ I_{\pi[i]} \geq 0 \qquad \forall\, i \in \{1..m\} \qquad \text{(LPIT 4)}$$

Equation (0) represents the sum of earliness and tardiness costs which has to be minimized. In equation (1) the completion time of the i^{th} job in π is computed by adding the processing time $q_{\pi[i]}$ and the idle time $I_{\pi[i]}$ to the completion time of the $(i-1)^{th}$ job in π. A fictitious completion time $C_{\pi[0]}$ is set to 0 in equation (3). In equation (2) earliness and tardiness are defined. By definition both are non-negative. If the completion time C_{fj} is greater than the due date d_{fj} then T_{fj} equals $(C_{fj} - d_{fj})$ and E_{fj} equals zero. If the completion time C_{fj} is smaller than the due date d_{fj} then E_{fj} equals $(d_{fj} - C_{fj})$ and T_{fj} equals zero. Due to the minimization of the objective function E_{fj} and T_{fj} will never both be greater than zero. In this way E_{fj} and T_{fj} will always get their proper values. Equation (4), finally, ensures non-negativity.

For any sequence $\pi(S)$ LPIT can be solved to get the optimally inserted idle time. However, as is recognized by Fry et al.[1987] and others (eg. Yano and Kim[1991]) an easy-to-understand algorithm exists to insert idle time optimally. This algorithm is, tailored to the particular problem, described in a number of versions, but the basic idea for all versions is the same and is based on the following observation: If a job is early the total costs may be decreased by right shifting this job[4]. For a job which is tardy right shifting the job will increase the total costs. For a subset of jobs in S for which no idle time is present between any two jobs of this subset, the net result of a right shift is determined by comparing the sums of decreases and increases.

[4] Right shifting a job means that the completion time of the job is increased. Equally, left shifting means that the completion time of a job is decreased.

Starting from the last job, the algorithm right shifts the jobs one by one, until either their due date is met or another subset of jobs is met. If the net result of further right shifting the new, enlarged, subset is positive, the whole subset is right shifted until either again a new subset is met, or until the net result of right shifting is changed (due to a job in the subset which meets its due date). If either the due date of the job is met, or the net result of right shifting the set of jobs becomes negative, the next job is taken, until all jobs have been done. The implementation of this algorithm as it is used here is based on Fry et al.[1987].

3.3 Sequencing jobs within a family

For jobs within one family, which all have the same processing time, it will be shown that it is always preferable to sequence them according to the earliest due date rule. This result is comparable with the result in Garey et al.[1988], for problems in which all tasks have the same length.

THEOREM 1

There is an optimal schedule in which, for all job types, jobs of the same job type are sequenced according to non-decreasing due dates.

PROOF[5]:

For interchanging two jobs of the same job type we do not need to consider setup costs or setup times. Therefore we can use a proof which is similar to the one used in Garey et al.[1988].

Suppose we are given a schedule with job J_j and job J_t belonging to the same job type, and job J_j being produced before job J_t while $d_j > d_t$. Possibly the start time of job J_t is not equal to the completion time of job J_j, either because first a number of other jobs have been scheduled, or because idle time has been inserted. Because job J_j and job J_t have equal processing times (they belong to the same job type) interchanging them leads to a feasible schedule.

For this situation (J_j precedes J_t, $d_j > d_t$), six cases can be distinguished,

[5] For ease of presentation, the index f for the family is left out of the notation since all jobs concern the same family.

namely both jobs early, i.e. $d_j > d_t \geq C_t > C_j$ (1), both jobs tardy, i.e. $C_t > C_j \geq d_j > d_t$ (2), and four cases in which only job t is tardy, i.e. $d_j \geq C_t \geq d_t \geq C_j$ (3), $d_j \geq C_t > C_j \geq d_t$ (4), $C_t \geq d_j \geq C_j \geq d_t$ (5) and $C_t \geq d_j > d_t \geq C_j$ (6). It can easily be checked that for the first two cases the costs for the schedule with job J_j and job J_t interchanged are equal to the costs of the original schedule, whereas for the other cases the costs for the schedule with job J_j and job J_t interchanged are lower than the costs of the initial schedule.

So, by using this argument repetitively, starting from any schedule we can always construct a schedule in which the jobs of the same job type are ordered according to their due dates, and which has costs lower than or equal to the costs of the original schedule. Since the ordering rule is transitive, this proves the theorem. □

Due to this theorem the number of sequences which have to be evaluated can be reduced significantly. An upper bound on the number of sequences can be given by $m!/(\Pi m_f!)$: the number of sub-sequences with respect to family f ($f \in \{1..F\}$) is restricted to 1 instead of $m_f!$. This is an upperbound, because the single sub-sequence for a family has to be partitioned in a number of batches. The feasible numbers of batches per family are mutually dependent. For instance, for a situation with two families it can not occur that family 1 consists of three batches, whereas family 2 consists of only one.

Still the number of potential sequences increases exponentially since a (weak) lower bound can be obtained as $\prod_{i=0}^{F-1} (F-i)^{m_{F-i}}$. In this expression it is assumed (without loss of generality) that $m_1 \geq m_2 \geq ... \geq m_F$. The expression is exponential in the (minimum) number of orders per family.

The result from the theorem in previous section can slightly be generalized to a problem in which the earliness costs and tardiness costs are job-dependent, ie. α and β are replaced by α_{fj} and β_{fj}. The processing times are still assumed to be equal for all jobs within a family. For this generalized problem it can be shown that if $\alpha_{fj} \leq \alpha_{ft}$ and $\beta_{fj} \geq \beta_{ft}$ for all j,t with $d_{fj} \leq d_{ft}$ then there is always an optimal sequence for which the jobs are ordered by a non-decreasing due date order.

A further generalization allows the processing times p_f to be job-dependent (p_{fj} for J_{fj}) too. When considering the sequencing of jobs within one family, this leads to a problem which is comparable with the problem considered in Fry et al.[1987]. The comparison of (adjacent) jobs within one family is now comparable to the comparison of two (adjacent) jobs in Fry et al.[1987], Yano and Kim[1991] or Garey et al.[1988]. Therefore, the dominance results from these articles are valid for orders within the same family in the generalized problem. We may also conclude that for this last generalized problem the sequencing of orders within one family is already NP-hard.

3.4 Finding the optimal solution

We have used two methods for finding the optimal solution of a given instance of the problem. Because of Theorem 1, it is assumed that for all product families f ($f\in\{1..F\}$) the jobs J_{fj} ($j\in\{1..m_f\}$) are numbered such that they are ordered according to their due dates. First we present a mixed integer programming formulation which can be solved by the use of commercially available software for generic mixed integer programming formulations. Next, we describe a branch and bound method which uses the structure of this particular problem. This branch and bound procedure has been implemented in Turbo Pascal® 6.0.

Mixed integer programming formulation for the static scheduling problem (IPS)

(Note : In equation 6, when summing over h and t, we have $h\in\{1..F\}$, $h\neq f$ and $t\in\{1..m_h\}$.)

$$\text{Minimize} \sum_{f=1}^{F} \sum_{j=1}^{m_f} \left[\alpha * E_{fj} + \beta * T_{fj} + \gamma * \tau_{fj} \right] \qquad \text{(IPS 0)}$$

subject to

$$T_{fj} - E_{fj} = C_{fj} - d_{fj} \qquad f\in\{1..F\},\ j\in\{1..m_f\} \qquad \text{(IPS 1)}$$

$$C_{f1} \geq p_f + s_f \qquad f\in\{1..F\} \qquad \text{(IPS 2)}$$

$$C_{fj} \geq C_{fj-1} + p_f \qquad f\in\{1..F\},\ j\in\{2..m_f\} \qquad \text{(IPS 3)}$$

$$C_{fj} \leq C_{ht} - p_h - s_h + M\phi_{fjht} \qquad f\in\{1..F\},\ h\in\{1..F\},\ f\neq h,\ j\in\{1..m_f\},\ t\in\{1..m_h\} \qquad \text{(IPS 4)}$$

$\phi_{fjht} + \phi_{htfj} = 1$ $\quad f \in \{1..F-1\},\ h \in \{f+1..F\},\ j \in \{1,..m_f\},\ t \in \{1..m_h\}$ (IPS 5)

$a_{fj} = \Sigma_{h \neq f} \Sigma_t \phi_{fjht}$ $\quad f \in \{1..F\},\ j \in \{1..m_f\}$ (IPS 6)

$\tau_{fj} \geq (a_{fj} - a_{fj-1})/N$ $\quad f \in \{1..F\},\ j \in \{2..m_f\}$ (IPS 7a)

$\tau_{f1} = 1$ $\quad f \in \{1..F\}$ (IPS 7b)

$T_{fj}, E_{fj}, C_{fj}, a_{fj} \geq 0\ ;\ \phi_{fjht}, \tau_{fj} \in \{0,1\}$ (IPS 8)

The basic idea behind this formulation is comparable to the formulation LPIT above, used to optimally insert idle time. It is extended, since the job sequence is not fixed anymore. The job is now determined by the zero-one variables ϕ_{fjht}, which indicate the precedence relations between jobs. Equations (6) and (7) have been added to determine the number of required setups.

In the objective function (0) the sum over all jobs J_{fj} of the costs of earliness, tardiness and setup is minimized. Equation (1) is equal to (LPIT 2) and determines earliness and tardiness. The inequalities (2) and (3) impose some restrictions on the completion times due to the completion times of other jobs of the same type. Further, if job J_{fj} is produced before job J_{ht} (inequalities (4), $\phi_{fjht}=0$) then the completion time of J_{fj} must be smaller than the completion time of job J_{ht} minus the processing time for job J_{ht} and minus the setup time for job J_{ht}. If job J_{fj} is produced after job J_{ht} ($\phi_{fjht}=1$) then the completion time of job J_{fj} is not restricted by the completion time of job J_{ht}. Equation (5) expresses that either job J_{fj} is produced before job J_{ht} or that job J_{ht} is produced before job J_{fj}. Note that the number of variables and constraints can be reduced by integrating equation (5) into the inequalities (4).

For any job J_{fj}, a_{fj} (in equation (6)) is the total number of jobs not of family f, that precede job J_{fj}. If the number of predecessors of another family is not equal for jobs J_{fj} and J_{fj-1}, then before job J_{fj} is produced a setup is required. In that case the zero-one variable τ_{fj} is restricted, by inequalities (7a), to a value greater then zero, so $\tau_{fj}=1$. If the number of predecessors is equal for J_{fj} and J_{fj-1} then τ_{fj} is not restricted, and because of the minimization of the objective function it will be $\tau_{fj}=0$. Obviously $\tau_{f1}=1$ for $f \in \{1..F\}$ (equation (7b)). Again, the number of variables and constraints may be decreased by integrating equation (6) into the inequalities (7).

It is easy to see that for problems with no setup costs (γ=0), equations (6) and (7) can be skipped. Since this leads to a reduction of the number of zero-one variables, solving these problems takes considerably less time than solving the problems with setup costs (γ>0).

The branch and bound method which has been used is comparable with the one described in Yano and Kim[1991]. They shortly describe a backward sequencing procedure (see also Hoogeveen and Van de Velde[1992]) for the problem without setup costs and setup times. For this problem, with setup costs and setup times, the same approach can be followed. Assumes that the root node of the search tree is at level 0. Then, for a set of m jobs, any node at level i ($i \in (0..m)$) in the search tree corresponds to a partial sequence which consists of the jobs in the final i positions. Due to Theorem 1 at any node only one job per family has to be considered as the next job to be added to the partial sequence.

A lower bound on the total costs for a node, at level i ($i \in (0..m)$) say, can be obtained as the sum of two partial lower bounds. The first partial lower bound is a lower bound on the costs for the i jobs in the partial sequence. The second partial lower bound is a lower bound on the costs for the (m-i) jobs not in the partial sequence. For the i jobs in the partial sequence the first partial lower bound can be obtained by computing the earliest possible starting time for these jobs. The earliest possible starting time can be obtained from the sum of the production times of the (m-i) jobs not in the partial sequence, increased by a minimum of setup time which is surely needed. Starting from this earliest possible starting time idle time can be inserted optimally. The costs of this partial schedule are a lower bound on the costs of the partial sequence when the full sequence is completed.

The second partial lower bound, for the (m-i) jobs not in the partial sequence, is computed as the corresponding minimum of setup costs which is sure to be incurred. As this lower bound is weak, a breadth first search strategy would lead to a high need for computer memory. Therefore, a depth first search strategy is used.

Despite its simplicity, the branch and bound method appears to be faster than solving the Mixed Integer Programming formulation by commercially available

software (see section 5).

4. Heuristic methods

Because the static scheduling problem is NP-hard, solving it optimally will become impossible for large instances, at least within a reasonable time. Since this problem has to be solved frequently in the simulation model a fast procedure is needed. Therefore, heuristic methods have to be exploited. In general, solutions produced by heuristic methods are not optimal. The goal here is to develop fast heuristics which on average produce results close to the optimum. In this section, a description of three heuristic methods for the solution of the static scheduling problem is given. Because of Theorem 1 all methods are constructed such that the solution found satisfies the condition that jobs of the same type are sequenced according to non-decreasing due dates.

Three methods build up a sequence by subsequently adding jobs, whereafter a schedule is created by optimally inserting idle time. These methods are the earliest due date rule, the clustered types/ shortest processing time rule, and the cheapest insertion rule. The fourth method is a local optimization method, used to improve the results of the above mentioned three methods.

4.1 The earliest due date procedure (EDD)

In this scheduling procedure, first a sequence is constructed by ordering all jobs, independent of their family, according to non-decreasing due dates. From this sequence a schedule is obtained by optimally inserting idle time. If demand for capacity is low, this procedure gives good results with respect to earliness and tardiness costs. However, when constructing a sequence it disregards setup costs. If the demand for capacity is low, efficiency is less important, and although costs are associated with the large number of setups, the due date performance (measured by earliness and tardiness costs) is very good in those cases. If efficiency is indispensable, due to high demand for capacity, the procedure can be expected to give bad results, due to the indirect effect of bad efficiency on the tardiness.

4.2 The clustered types/shortest processing time procedure (SPT)

Instead of focussing on the earliness and tardiness costs this procedure minimizes the number of setups. First sub-sequences are constructed by combining all jobs of the same type (using an EDD order because of Theorem 1). Then the sub-sequences are ordered according to increasing p_f, the production time for a single job of family f. The schedule is obtained from this sequence by optimally inserting idle time.

This procedure minimizes the number of setups, disregarding the earliness and the tardiness costs. In contrast with the EDD procedure, this procedure can be expected to perform well when demand for capacity is high. In those cases it is highly necessary to be efficient. If the demand for capacity is low, however, efficiency is less needed and the SPT procedure can be expected to show a bad performance, since the earliness and tardiness costs will be high.

4.3 The cheapest insertion procedure (CI)

The procedure consists of two steps. The first step is an initial step in which the first job is added to the schedule. In the second, iterative, step the other jobs are added one by one.

In the initial step compute for all jobs the costs of scheduling just this one job. Choose the job which adds minimum costs. In the second step compute for all remaining jobs the best position to be inserted at in the current schedule. Choose the job which adds minimum insertion costs, and insert it at the computed best position. In both steps, in case of ties any job may be chosen. In our implementation, we have chosen the one with the smallest index. Repeat the second step until all jobs are scheduled.

In contrast with the previous four procedures, which construct a sequence without making explicit calculations, this procedure does calculate the costs of the schedule. Thereby it computes earliness and tardiness costs as well as setup costs. It is one variant of a number of so-called insertion procedures, which all construct a schedule by inserting jobs one by one in the schedule at the most profitable place. In this case the choice for a particular job is made by taking the job which allows for the cheapest insertion.

4.4 Local optimization procedure (LOC)

In general, local optimization methods, also called local search methods, try to improve on an existing schedule. This is done by defining a neighborhood relation among schedules. Two schedules are neighbors if they can be obtained from each other by carrying out exactly one operation. As an example of an operation consider the exchange of two adjacent jobs. An algorithm starts with any feasible schedule and searches among all neighbors of the schedule for the best neighbor (in terms of the value of the objective function). If a better neighbor is found, the current schedule is replaced by this neighbor. The procedure is repeated until a schedule is obtained which only has neighbors with worse values of the objective function.

We have used a local optimization method in which the neighborhood relation is defined by two operations. The first operation is a pairwise interchange between two adjacent jobs, see Figure 1(a+b). For the second operation we consider a schedule as an ordered number of batches. This operation consists of a backward part and a forward part. Both parts are leading to a neighbor. The backward (forward) part of this operation moves the first (last) job of a batch to the last (first) position in the previous (next) batch of the same type. See Figure 1(a+c). As mentioned, the three constructive heuristic procedures from sections 4.1, 4.2 and 4.3 are used to obtain a starting schedule.

In the simulation models a combination of these three heuristic procedures, all improved by local optimization, is used. The scheduling procedure in the simulation procedure simply computes all three, after which the best schedule is picked. The heuristic procedures used in this combination can be seen as complementary: The EDD procedure considers the earliness and tardiness costs only, while the SPT procedure minimizes the setups costs, but does not consider earliness and tardiness. Although an SPT schedule is often very bad, especially in those cases where EDD may not do well it often finds good solutions. The CI procedure considers the whole cost function at once.

5. Results

For a number of representative examples the solutions of the heuristic procedures

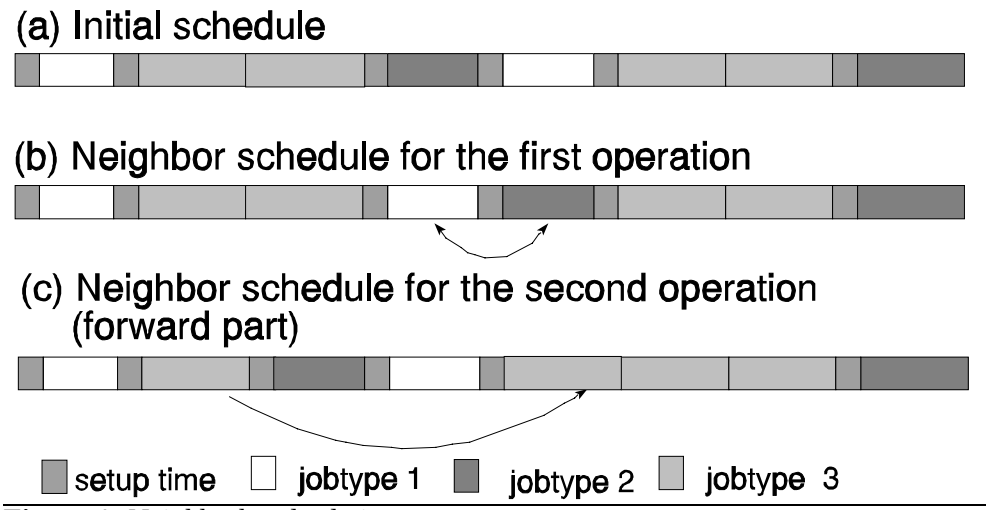

Figure 1. Neighborhood relation.

in section 4 can be compared with the optimal solutions. Thus, insight can be obtained in the performance of both the individual heuristic methods and the combined heuristic method used in the simulation model. Either with the mixed integer programming formulation or with the problem specific branch and bound method the optimal schedule is computed for a number of instances. The results are then compared to the results of the heuristic methods.

Problems are tested for a number of combinations of the parameters F and m. For all problems the processing times p_f ($f \in \{1..F\}$) have been chosen randomly from the integer uniform distribution between 1 and 6. All setup times s_f ($f \in \{1..F\}$) have been fixed at the value 2. Given the processing and setup times, the due dates have been determined randomly, using a Poisson distribution with the arrival intensity parameter $\lambda = 1/\theta * (\bar{p} + \bar{s})$, where $\bar{p}$ is the average of the processing times and $\bar{s}$ is the average of the setup times over al families.

This way the average time between the due dates is related to the average time needed for setup and production of one job. The parameter θ was used to distinguish between problems with a heavy load ($\theta = 0.8$) and problems with a small load ($\theta = 1.5$). For all combinations of F and m we generated 20 problems with $\theta = 0.8$ and 20 problems $\theta = 1.5$. All jobs were randomly assigned to one of the families, all

families having a chance of 1/F for being picked. For all problems four cost settings are used. For all settings the earliness costs are fixed at value α=1, whereas the tardiness costs β and the setup costs γ are varied. The four settings are (α,β,γ)=(1,8,0), (α,β,γ)=(1,8,20), (α,β,γ)=(1,8,40) and (α,β,γ)= (1,15,20).

For all combinations of F,m and θ, for all costs settings, we computed the average over the 20 problems of the relative deviations from the optimum value. Given the objective value x of the heuristic solution and the optimal value opt this was computed as

$$\frac{x - opt}{opt} * 100\%$$

Comparing the computation times, the branch and bound method appears to be much faster solving the mixed integer programming formulation by using mixed integer programming software[6]. Using a personal computer (486DX at 40Mhz) for the mixed integer programming formulation the smaller problems ($m \leq 8$) were solved within a few seconds and almost all problems were solved within 2 minutes. The branch and bound method solved these problems in less than half a second.

For some of the larger problems ($m \geq 12$) the computation times became far too large using the mixed integer programming formulation. By computing only the solutions for γ=0 for these problems we could delete equations 6 and 7, and the variables a_{fj} and τ_{fj} from the mixed integer programming formulation in section 3. Because of this, the number of zero-one variables was reduced, which led to a severe reduction of the computation times. However, the branch and bound method could easily solve all these larger problems. For problems of size F=4, m=16, including setup costs, the average computation time was about 20 seconds.

In the appendix (see page 14) the results are presented. As can be seen, in all cases under consideration the combination of the earliest due date method with the local optimization method (EDD-LOC) performs good. This is in contrast with clustered type/shortest processing time method, which on average gives bad results, even after improvement with the local optimization method (SPT-LOC). The

[6] We used the XA software package from Sunset Software Technology.

cheapest insertion method (CI) appears to give good results all by itself. However, the improvement by local optimization is modest, compared to the improvements of the solutions found by the earliest due date method. Anyhow, local optimization seems to perform quite well starting from any method.

Comparing the results for the different costs settings, it appears that the earliness/ tardiness oriented EDD method performs better when γ is smaller. This corresponds to what may be expected. The results of the clustered types/shortest processing time method (SPT), which is oriented on setup costs, also correspond to the expectations. Obviously, with $\gamma=0$ the results are very poor. As γ grows the results get better. Still, however, the results for this method are moderate. With respect to the different cost settings, the power of the local optimization method is shown once more. The effects of varying the cost parameters disappear after using the local optimization method.

Note that the moderate results of the clustered types/shortest processing time rule can not be improved within reasonable margins by the local optimization method. The main reason for this is that the local optimization method has difficulties to break up a large batch into two smaller ones, since it needs more than one operation to do so.

In the last row (BOA) we show the results when taking for all cases the best solution from all heuristic methods. This is the heuristic method used to create the schedules in the simulation model. From this we learn that this hybrid method almost always provides a solution which is within 2% of the optimal solution, which is satisfying.

6. Discussion

This paper has described a scheduling problem which combines earliness costs, tardiness costs and both setup times and setup costs. Two methods for finding the optimal solution of this scheduling problem have been presented and a number of heuristic methods were described and tested for a number of different cost settings. Using a simple local optimization method the results obtained are almost always within a few percent of the optimum. Also, it appears that in almost all cases at least

one of the heuristic methods produces the optimal solution. Therefore, taking a combination of a few heuristic methods, that are complementary by nature, may yield near-optimal solutions for all cases.

With respect to further research two main suggestions can be given. The first suggestion concerns the generalized static scheduling problem. In the most general formulation of the problem both the processing times and the costs parameters are allowed to vary per job. How should the mixed integer programming formulation and the branch and bound method be adapted in order to find optimal solutions for this generalized problem? It is no longer optimal to choose a due date ordering within the family. Furthermore, one may ask what a good heuristic method looks like. An interesting suggestion may be to stick to a due date ordering in a first stage, using heuristics like the ones presented here, and thereafter optimize the ordering within the batches of jobs of the same family.

The second suggestion for further research concerns the behavior of the proposed scheduling heuristics in a more dynamic setting. In practice the set of jobs to be scheduled changes continuously. How should we cope with the arrival of new jobs? Do they need to be scheduled immediately or is it sufficient to reschedule only periodically? In the last case, what should be the length of the rescheduling period? One may also ask what happens when the set of jobs to be scheduled grows larger and larger. Do we still need to schedule all jobs at once, or is it possible to restrict our horizon, and use a rolling schedule. If so, how large should we take our horizon and what will be the review period?

References

Baker, K.R. & G.D. Scudder (1990), Sequencing with earliness and tardiness penalties : A review, Operations research, vol. 38, no. 1.

Coleman, B.J. (1992), Technical note : A simple model for optimizing the single machine early/tardy problem with sequence-dependent setups, Production and operations management, vol. 1, no. 2, pp. 225-228.

Fry, T.D., R.D. Armstrong & J.H. Blackstone (1987), Minimizing weighted absolute deviation in single machine scheduling, IIE transactions, vol. 19, no. 4.

Garey, M.R., R.E. Tarjan & G.T. Wilfong (1988), One processor scheduling with symmetric earliness and tardiness penalties, Mathematics of operations research, vol. 13, no. 2.

Gupta, S.K. & J. Kyparisis (1987), Single machine scheduling research, Omega, vol. 15, no. 3.

Hoogeveen, J.A. & S.L. van de Velde (1992), Minimizing total inventory cost on a single machine in just-in-time manufacturing, In : J.A. Hoogeveen, Single-Machine bicriteria scheduling, Ph.D. thesis, CWI, Amsterdam, the Netherlands.

Jordan, C. & A. Drexl (1994), Lotsizing and scheduling by batch sequencing, Manuskripte aus den Instituten for Betriebswirtschafslehre der Universität Kiel, nr. 343.

Monma, C.L. & C.N. Potts (1989), On the complexity of scheduling with batch setup times, Operations research, vol. 37, no. 5.

Ow, P.S. & T.E. Morton (1989), The single machine early/tardy problem, Management Science, Vol. 35, No. 2.

Potts, C.N. & L.N. van Wassenhove (1992), Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity, Journal of the operations research society, vol. 43, no. 5.

Ten Kate, H.A. (1994), Towards a better understanding of order acceptance, International journal of production economics, vol. 37, pp. 139-152.

Woodruff, D.L. & M.L. Spearman (1992), Sequencing and batching for two classes of jobs with deadlines and setup times, Production and operations management, vol. 1, no. 1.

Yano, C.A. & Y.-D. Kim (1991), Algorithms for a class of single-machine weighted tardiness and earliness problems, European journal of operational research, vol. 52.

Zdrzalka, S. (1991), Approximation algorithms for single-machine sequencing with delivery times and unit batch setup times, European journal of operational research, vol. 51.

Appendix 1 Test results for the static scheduling problem.

Average relative deviations (%) for F=2, m=8.

Procedure	θ=0.8			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	21.48	26.25	36.47	27.00
CI	4.92	12.24	14.38	14.26
SPT	207.49	40.10	25.02	46.25
EDD-LOC	1.18	2.39	1.03	1.09
CI-LOC	1.02	1.05	2.18	0.90
SPT-LOC	5.16	1.03	2.74	1.53
BOA	0.00	0.00	0.00	0.00

Procedure	θ=1.5			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	8.40	29.90	42.23	30.85
CI	2.03	9.68	9.27	10.97
SPT	1989.47	138.04	69.63	190.83
EDD-LOC	0.03	0.42	0.85	0.36
CI-LOC	0.00	0.40	2.85	0.31
SPT-LOC	15.73	0.91	2.03	0.74
BOA	0.00	0.40	0.00	0.31

Average relative deviations (%) for F=2, m=16.

Procedure	θ=0.8			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	57.38	68.85	82.93	70.56
CI	16.75	23.23	20.86	18.95
SPT	1008.46	188.72	114.50	233.36
EDD-LOC	0.62	1.95	3.83	1.83
CI-LOC	1.02	1.24	5.45	1.38
SPT-LOC	38.83	17.86	9.87	8.84
BOA	0.00	0.00	1.35	0.00

Procedure	θ=1.5			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	22.76	35.57	52.16	36.44
CI	2.44	12.87	20.21	13.45
SPT	7950.76	497.20	284.11	601.32
EDD-LOC	4.54	1.39	1.54	2.11
CI-LOC	0.89	2.05	5.57	2.01
ESP-LOC	4.54	1.39	1.54	2.49
BOA	0.00	0.24	0.00	0.24

Average relative deviations (%) for F=3, m=6.

Procedure	θ=0.8			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	24.67	26.98	30.61	27.44
CI	3.51	5.04	8.50	5.36
SPT	465.91	71.67	43.53	95.73
EDD-LOC	0.34	0.00	0.00	0.00
CI-LOC	0.00	0.19	0.18	0.05
SPT-LOC	2.67	3.42	3.20	3.13
BOA	0.00	0.00	0.00	0.00

Procedure	θ=1.5			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	11.05	14.37	19.61	14.35
CI	3.06	4.85	3.27	8.29
SPT	870.86	77.38	37.07	99.97
EDD-LOC	0.60	0.21	0.86	1.25
CI-LOC	0.74	1.70	0.44	1.14
SPT-LOC	16.40	2.70	1.49	2.55
BOA	0.60	0.00	0.00	0.00

Average relative deviations (%) for F=3, m=12.

Procedure	θ=0.8			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	77.43	70.69	74.83	80.09
CI	31.45	20.16	17.28	33.44
SPT	383.06	107.57	65.00	139.25
EDD-LOC	5.31	2.88	3.46	1.65
CI-LOC	6.25	6.52	5.59	7.96
SPT-LOC	23.06	17.74	17.85	21.43
BOA	0.34	0.93	0.92	0.94

Procedure	θ=1.5			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	13.96	24.26	35.39	24.66
CI	1.66	15.34	16.01	15.21
SPT	2797.81	309.90	176.75	399.68
EDD-LOC	0.87	1.79	3.90	0.93
CI-LOC	0.00	4.88	7.81	5.06
SPT-LOC	65.81	12.76	7.54	13.84
BOA	0.00	0.09	1.36	0.09

Average relative deviations (%) for F=4, m=8.

Procedure	θ=0.8			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	33.22	33.71	38.11	35.60
CI	13.55	8.77	8.63	11.82
SPT	473.86	67.19	41.69	87.94
EDD-LOC	0.15	0.12	1.25	0.78
CI-LOC	2.71	2.57	3.09	3.09
SPT-LOC	17.95	3.91	2.66	4.87
BOA	0.00	0.07	0.30	0.64

Procedure	θ=1.5			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	12.39	15.12	23.89	14.92
CI	3.62	7.52	5.41	7.75
SPT	2630.89	147.71	85.07	175.42
EDD-LOC	1.19	0.89	1.11	0.66
CI-LOC	2.47	2.39	1.01	2.81
SPT-LOC	4.70	6.54	7.66	7.22
BOA	0.67	0.16	0.41	0.00

Average relative deviations (%) for F=4, m=16.

Procedure	θ=0.8			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	68.26	56.50	61.82	66.19
CI	28.01	17.91	24.81	21.66
SPT	633.21	163.61	99.48	209.40
EDD-LOC	4.64	3.29	4.29	4.11
CI-LOC	1.61	7.13	5.59	7.43
SPT-LOC	39.02	8.55	8.59	20.71
BOA	1.36	1.48	1.59	2.16

Procedure	θ=1.5			
	α,β,γ 1,8,0	α,β,γ 1,8,20	α,β,γ 1,8,40	α,β,γ 1,15,20
EDD	19.86	22.38	31.83	24.37
CI	4.71	14.61	20.07	15.72
SPT	6675.40	274.13	150.58	321.89
EDD-LOC	1.84	0.69	2.55	0.78
CI-LOC	1.75	6.07	10.15	5.69
SPT-LOC	231.35	47.40	27.59	56.30
BOA	0.93	0.53	1.28	0.66