International Journal of Operations Research

International Journal of Operations Research Vol. 3, No. 2, 144–154 (2006)

Parallel Machine Scheduling by Family Batching with Sequence-independent Set-up Times

H.A.J. Crauwels[1,*], P. Beullens[2], and D. Van Oudheusden[3]

[1]Hogeschool voor Wetenschap & Kunst, De Nayer instituut, J. De Nayerlaan 5, B-2860 Sint-Katelijne-Waver, Belgium

[2]University of Portsmouth, Department of Mathematics, Buckingham Building, Lion Terrace, Portsmouth PO1 3HE, England

[3]K.U.Leuven, Centre for Industrial Management, Celestijnenlaan 300A, B-3001 Leuven, Belgium

Received November 2005; *Revised June 2006*; *Accepted July 2006*

Abstract—This paper presents a new approach for transforming MRP orders, planned periodically, e.g. on a weekly base, into a detailed schedule of jobs. In this model for a parallel machine environment, the jobs are partitioned into families and a family specific set-up time is required at the start of each period and of each batch, where a batch is a maximal set of jobs in the same family, that are processed consecutively. An integer program is formulated for both the problem of minimising the number of overloaded periods and the problem of minimising the total overtime. These programs generate benchmark results for the heuristic approach. A heuristic model is developed that constructs a schedule in which overloaded periods are relieved and set-up time is saved. In this approach, the job sequence is constructed by repeatedly solving a knapsack problem for each machine. The weights used in this knapsack problem relate to the preferred priorities of the jobs not yet scheduled and determine the quality of the final sequence. The different features of the heuristic model are compared using a large set of test problems. The results show that the quality of the final schedule depends on an appropriate choice of the weights.

Keywords—Set-up times, Family batching, Parallel machine scheduling

1. INTRODUCTION

In an MRP II environment, the MRP module generates a set of planned orders, each having a release date and a due date. Because MRP assumes an infinite capacity, there is a need for detailed scheduling in order to construct a feasible schedule. The quality of the resulting schedule can be evaluated using a number of criteria, for example: meeting due dates, minimising lead time, minimising in-process-inventory, minimising total set-up time or maximising machine utilisation. The responsability of specifying when and how the planned orders (or jobs) are to be processed is delegated to the shop floor scheduler. There are several possible ways to derive the detailed schedule, depending on the objectives the scheduler has in mind.

In many practical situations, the jobs to be processed are divided into different families such that jobs of different families require different sets of processing facilities. In such problems, a set-up time is necessary for rearranging processing facilities whenever there is a switch from processing a job in one family to another job from a different family. In general, these set-up times are sequence dependent. As a consequence, decisions about the sizes of the production batches and the sequencing of the batches have to be taken simultaneously. Most research has been dedicated to single machine problems. Several results can be found in survey papers, e.g. Webster and Baker (1995), Drexl and Kimms (1997), and Potts and Kovalyov (2000). A comprehensive review of scheduling problems involving set-up considerations is provided by Allahverdi et al.(1999). In this review scheduling problems are classified into batch and non-batch, and sequence-independent and sequence-dependent set-ups. Since the appearance of that survey paper, there has been an increasing interest in scheduling problems with set-up times or costs. An up-to-date survey is conducted by Allahverdi et al. (2006).

When a large number of jobs have to be processed, several parallel machines can be used. In this situation, also the assignment of the production batches to single machines has to be determined. Quadt and Kuhn (2000) describe for example the back-end assembly operations of the semiconductor manufacturing process, which are die attach, wire bonding and molding. The set-up times are relatively long between product families (up to 12 hours) and are in reality sequence dependent. But the degree of sequence-dependency is modest, and therefore Quadt and Kuhn use average product family set-up times which are

* Corresponding author's e-mail: hcr@denayer.wenk.be

sequence independent.

Chen and Powell (2003) consider two particular parallel machine problems with multiple job families: minimising total weighted completion time and minimising the number of weighted tardy jobs. They propose exact solution algorithms based on column generation. Their computational experiments show that these algorithms are capable of solving problems with up to 40 jobs on up to six machines within reasonable computation time. For the total weighted completion time problem, Dunstall and Wirth (2005) develop a number of simple, effective and efficient heuristic methods based on ideas that have appeared previously in the literature for the analogous single machine problem, the parallel machine problem without set-ups and the parallel machine makespan problem with set-ups.

Several research papers about lotsizing on parallel machines are inspired by a technical report from Chesapeake Decision Sciences, Inc. Baker and Muckstadt (1989) provide in this report the CHES problems, a set of problem instances, based on existing business problems. These problems comprise parallel production lines and sequence-dependent set-up costs, but no set-up times. The objective is to find a production schedule that minimises the sum of set-up, production and holding costs minus sales revenue taking into account the product demand and the capacity of the machines. Kang et al. (1999) designed the sequence splitting model for solving the CHES problems. This model splits an entire schedule into subsequences, leading to tractable subproblems. Column generation and branch and bound are the basic elements of the heuristic solution method.

Meyr (2002) proposes a combination of local search metastrategies with local reoptimization for solving the problem of simultaneous lotsizing and scheduling on non-identical parallel machines. In his model sequence-dependent set-up times are integrated as a further reduction of the limited capacity of the machines. The solution procedure is quite flexible, because - after some minor modifications - it can solve the CHES problems.

Clark (2003) develops models for constructing a capacity feasible master production schedule (MPS) in material requirements planning (MRP) systems. The exact MIP model minimises the total costs associated with stocks and backorders and takes into account sequence-dependent set-up times. It is optimally solvable only for small product structures. He proposes an approximate model and solution method in which set-ups and lots are scheduled on a period-by-period basis. This model is able to schedule set-ups of up to 100 products on 10 machines over 5 periods in reasonable computing time.

In this paper, the method proposed by Crauwels and Van Oudheusden (2003), for transforming planned MRP orders into a detailed schedule in a single machine environment, is extended to the parallel machine environment. We assume that the planning horizon is segmented into a finite number of time buckets of equal length, e.g. into weeks. The end times of these periods constitute the different due dates of the jobs. The jobs are divided into a number of families: a sequence-independent set-up time is incurred between jobs of different families and whenever a job is the first to be processed in a period. A job has to be processed (inclusive a specific set-up of its own) without interruption in such a period.

When all jobs are scheduled in the single period just before their common due date, there is a large probability that the resulting sequence is not feasible. Some periods will have idle time and in other periods there will be not enough capacity to process all the scheduled jobs. Thus, the jobs from the overloaded periods should be shifted to the underloaded periods. Probably, several jobs may be worth being considered for shifting. A more interesting job is a job that saves set-up time, relieves overloaded periods and helps to maximise the utilisation of the resources.

The main objective of this study is to adapt a number of simple heuristic rules that were developed for the single machine case so that they can be conveniently applied by a dispatcher in a parallel machine environment. The constructed schedules can be evaluated on different performance criteria. And, because different circumstances in the workshop require different performance criteria to be optimised, we also investigate the relation between a specific rule and a specific criterion. In addition, two integer programming models are formulated, mainly for using their outcomes as benchmarks. They are probably not very useful in practice because of their complexity and the fact that, at one time, only a single objective can be optimised.

On the one hand our approach is inspired by the research carried out in the area of scheduling with family batching. In this context, the motivation for batching jobs is a gain in efficiency: it may be cheaper or faster to process jobs in a batch than to process them individually. On the other hand, our technique is also related to lotsizing: grouping together planned MRP orders of consecutive periods in order to minimise the sum of inventory and set-up costs (Maes and Van Wassenhove, 1988). One of the first reviews about the integration of scheduling with batching and lotsizing is presented by Potts and Van Wassenhove (1992).

In Section 2, we give a formal statement of our problem and Section 3 introduces the integer programming models. In Section 4, the new heuristic approach is described. Section 5 reports on computational experience and some concluding remarks are contained in Section 6.

2. PROBLEM FORMULATION

To state our scheduling problem more precisely, we are given N jobs divided into F families. Each family f, for $1 \leq f \leq F$, contains n_f jobs. The jobs are numbered 1, 2, …, N. Sometimes it is more convenient to refer to job (i, f), which is the ith job in family f, for $1 \leq i \leq n_f$. There are M identical parallel machines available on which these jobs can be scheduled. We let p_{if} denote the processing time of job (i, f). The planning horizon is segmented into a finite number (T) of equal length time (L) periods. Each job has a release date r_{if}, the moment it becomes available

for processing, which corresponds to the start time of such a period. Each job has a due date d_{ij} which is the end time of such a period. There are at most T different end times and sometimes we refer to them as D_t with $1 \leq t \leq T$. In a single machine environment, all jobs in one family can be assumed to have different due dates. If there are two or more jobs of some family with the same due date, they can in many practical situations be considered as one large job that requires only one set-up. In a parallel machine environment several jobs of a family with the same due date can be scheduled on different machines each requiring a set-up. In both cases jobs are labeled in order of non-decreasing due dates d_{ij}. A sequence-independent set-up time s_f is incurred whenever a job in family f is processed immediately after a job of a different family. Also, an initial set-up time s_f is required if a job from family f is the first to be processed in a period. In addition, the complete processing of a job, possibly preceded by a job specific set-up has to be carried out in one period.

Note that in this formulation real set-up times are used and not set-up costs as is usually the case in more classical lotsizing models. We believe that set-up time is a much more practical concept. As long as costs (for set-up, overtime, and inventory) are directly proportional to their respective use of time, the approach will also find cost-efficient solutions. In certain situations, however, costs may not be proportional to time and then the application of the presented approach should be done with caution. The cost, for example, could be high for switching between certain families even though the changeover time is relatively small (Allahverdi et al., 1999), or vice versa. Likewise, the cost of keeping inventory for one additional period may be relatively high for certain families and small for other families. Finally, the cost of production of certain families may be relatively high compared to their production time when, for example, more or higher skilled operators are needed.

In a first attempt for solving a parallel machine scheduling problem, the capacity of the M parallel machines can be taken together. The resulting virtual single machine problem can be solved with single machine techniques and the sequence of jobs has then to be subdivided across the different parallel machines. In our problem however, this approach would be too simplistic because of the family set-up times. During the last phase, when the jobs have to be subdivided across the different machines, additional set-up times would have to be incorporated. Probably, this would take too much time of the available machine capacity.

3. INTEGER PROGRAMMING FORMULATIONS

By defining variables

$$x_{ijtk} = \begin{cases} 1, & \textit{job } (i, j) \textit{ is processed in period } t \textit{ on machine } k, \\ 0, & \textit{otherwise}, \end{cases}$$
$$(i,j) \in \boldsymbol{N},\ t \in \boldsymbol{T},\ k \in \boldsymbol{M},$$

$$y_{ftk} = \begin{cases} 1, & \textit{a set-up for family } f \textit{ occurs in period } t \textit{ on machine } k, \\ 0, & \textit{otherwise}, \end{cases}$$
$$f \in \boldsymbol{F},\ t \in \boldsymbol{T},\ k \in \boldsymbol{M},$$

$$z_{tk} = \begin{cases} 1, & \textit{period } t \textit{ on machine } k \textit{ is overloaded}, \\ 0, & \textit{otherwise}, \end{cases}$$
$$t \in \boldsymbol{T},\ k \in \boldsymbol{M},$$

with $\begin{cases} \boldsymbol{F} = \{1,\ldots,F\} \\ \boldsymbol{N} = \{(1,1),\ldots,(n_1,1),(1,2),\ldots,(n_f,F)\} \\ \boldsymbol{T} = \{1,\ldots,T\} \\ \boldsymbol{M} = \{1,\ldots,M\} \end{cases}$

we obtain the following formulation for minimising the number of overloaded periods:

$$\text{minimize} \quad \sum_{k \in M} \sum_{t \in T} z_{tk} \tag{1}$$

subject to

$$\sum_{k \in M} \sum_{t \in T} x_{ijtk} = 1,\ (i,j) \in \boldsymbol{N}, \tag{2}$$

$$x_{ijtk} \leq y_{jtk},\ (i,j) \in \boldsymbol{N},\ t \in \boldsymbol{T},\ k \in \boldsymbol{M}, \tag{3}$$

$$\sum_{f \in F} s_f y_{ftk} + \sum_{(i,f) \in N} p_{ij} x_{ijtk} \leq L + Q z_{tk},\ t \in \boldsymbol{T},\ k \in \boldsymbol{M}, \tag{4}$$

$$x_{ijtk} = 0 \ \text{ if } tL \leq r_{ij} \ \text{ or } tL > d_{ij},\ (i,j) \in \boldsymbol{N},\ k \in \boldsymbol{M}, \tag{5}$$

$$x_{ijtk} \in \{0,1\},\ (i,j) \in \boldsymbol{N},\ t \in \boldsymbol{T},\ k \in \boldsymbol{M}, \tag{6}$$

$$y_{ftk} \in \{0,1\},\ f \in \boldsymbol{F},\ t \in \boldsymbol{T},\ k \in \boldsymbol{M}, \tag{7}$$

$$z_{tk} \in \{0,1\},\ t \in \boldsymbol{T},\ k \in \boldsymbol{M}. \tag{8}$$

Constraints (2) ensure that each job is processed in exactly one period on one machine. Constraints (3) ensure that the necessary set-ups are executed: if job (i, f) is processed in period t, a set-up for family f is required in period t. A capacity restriction has been imposed on each machine and on each period by constraints (4): the required set-ups plus the processing times must be less than the length of the period (L), except for the overloaded periods where this length is increased by a term Q. Constraints (5) ensure that each job is processed between its release date and its due date.

When Q in constraints (4) is set to a very large value, the minimisation of the number of overloaded periods often results in an unrealistic schedule where all the overload is assembled into just one period on one specific machine. In practical situations, a restriction on the magnitude of the overload is imposed. For the computational experiments, we choose to set Q equal to the length of a period (L). For some problem instances, the time window established by

constraints (5) is rather narrow. Therefore, it is possible that because of the limited overtime, no feasible solution can be calculated for the model. By taking a rather large value for Q, we eliminate this phenomenon as much as possible.

By changing the variables z_{tk}, with $t \in \boldsymbol{T}$ and $k \in \boldsymbol{M}$, from binary to positive real and by replacing the capacity constraints (4) with

$$\sum_{f \in \boldsymbol{F}} s_f y_{ftk} + \sum_{(i,f) \in N} p_{if} x_{iftk} \leq L + z_{tk}, \quad t \in \boldsymbol{T}, \ k \in \boldsymbol{M}, \tag{9}$$

we formulate a mixed integer programme for minimising the total overtime. In this model the overload will be spread more uniformly across the different time periods and machines.

4. A HEURISTIC APPROACH

The idea is to include some jobs of the following periods in the current period (with due date D_t) in order to realise a feasible schedule, to save set-up time and to have fully utilised resources.

First, the jobs that have to be processed before the end of the period because of their due date, are included in the current period. During the remaining time, some jobs of the following periods can be added and these are determined by solving a knapsack problem. For each of these jobs i, from some family f, a weight w_{if} is defined that is related to the performance criteria mentioned above. For example, a job gets a larger weight when it is a job from a family with a large set-up time and especially when its family contains only a few jobs. In addition, jobs from overloaded periods have to be given preference. On the other hand, care must be taken to process a job not too early.

To describe our heuristic method, let S_t be the set of jobs not yet scheduled at the beginning of period t. For each period $t = 1, 2, \ldots, T$:

- define a subset of jobs not yet scheduled: $\boldsymbol{J}_t = \boldsymbol{S}_t \cap \{(i,f) \mid d_{if} = tL\}$, and include the jobs of $\boldsymbol{J}_t$ in the current period by setting $x_{iftk} = 1$ and $y_{ftk} = 1$ for some machine k;
- when the current period t leaves idle time on some machine k, i.e. $\sum_{f \in \boldsymbol{F}} s_f y_{ftk} + \sum_{(i,f) \in \boldsymbol{J}_t} p_{if} x_{iftk} < L$, define a set of additional candidates $\boldsymbol{K}_t \backslash \boldsymbol{J}_t$, with $\boldsymbol{K}_t = \boldsymbol{S}_t \cap \{(i,f) \mid r_{if} \leq (t\text{-}1)L\}$, and calculate a weight w_{if} for each of these candidates;
- add some jobs of $\boldsymbol{K}_t \backslash \boldsymbol{J}_t$ (jobs of the following periods) to the current period t by solving a knapsack-like problem:

$$\text{maximize} \quad \sum_{k \in \boldsymbol{M}} \sum_{(i,f) \in \boldsymbol{K}_t \backslash \boldsymbol{J}_t} w_{if} x_{iftk} \tag{10}$$

subject to

$$x_{iftk} \leq y_{ftk}, \ (i,f) \in \boldsymbol{K}_t, \ k \in \boldsymbol{M}, \tag{11}$$

$$\sum_{f \in \boldsymbol{F}} s_f y_{ftk} + \sum_{(i,f) \in \boldsymbol{K}_t} p_{if} x_{iftk} \leq L, \ k \in \boldsymbol{M}, \tag{12}$$

$$x_{iftk} \in \{0, 1\}, \ (i,f) \in \boldsymbol{K}_t, \ k \in \boldsymbol{M}, \tag{13}$$

$$y_{ftk} \in \{0, 1\}, \ f \in \boldsymbol{F}, \ k \in \boldsymbol{M}. \tag{14}$$

Constraint (12) ensures that the total set-up plus processing times of the jobs in the sequence is smaller than the available capacity in the current period for each machine.

Problem (10)-(14) is a 0-1 multiple knapsack problem with the extra complication of set-up times. Martello and Toth(1990) suggest a bound-and-bound algorithm in which the upper bound is based on the standard surrogate relaxation. In this relaxation, the multiple knapsacks are replaced by one large knapsack

$$\sum_{f \in \boldsymbol{F}} s_f y_{ft} + \sum_{k \in \boldsymbol{M}} \sum_{(i,f) \in \boldsymbol{K}_t} p_{if} x_{iftk} \leq M \times L \tag{15}$$

with the binary variables y_{ft} indicating whether there has to be a set-up in period t for family f or not. Consequently, constraints (11) are changed to $x_{iftk} \leq y_{ft}$. The resulting problem is solved with a standard zero-one knapsack problem algorithm. However, testing the feasibility of the upper bound is an NP-complete problem. Therefore, they require in addition a good lower bounding heuristic method, in which M individual 0-1 knapsack problems are solved. Pisinger (1999) has developed a procedure for validating the feasibility of the upper bound by solving a series of subset-sum problems. The resulting algorithm performs quite well for large multiple knapsack problems.

Because of the extra complicaton of the set-up times the upper bound obtained by (15) is rather weak. Only one set-up time is taken into account for each family. In order to construct a feasible solution, additional set-up times have to be incorporated when two or more jobs of the same family are scheduled within the considered period but on different machines. Therefore, we suggest the following approach that is executed for each period $t = 1, 2, \ldots, T$. First, we assign the jobs that have to be scheduled before the end of the period (subset $\boldsymbol{J}_t$) to a specific machine. This method is based on the longest processing time (LPT) rule. A priority list is built by ordering the jobs of subset $\boldsymbol{J}_t$ according to non-increasing processing times. The jobs are scheduled in this order, each time assigning a job to the machine with the least amount of processing already assigned. In addition, once a job is scheduled, it is checked whether there are some jobs of the same family in subset $\boldsymbol{J}_t$. If this is the case and there is still enough time available on that machine, these jobs are also added to the sequence of that machine. With this rule, the processing load is more or less equally spread among the different machines. Then, we include a number of additional jobs of following periods (subset $\boldsymbol{K}_t \backslash \boldsymbol{J}_t$) into the current period by solving a 0-1 knapsack problem for each machine k:

$$\text{maximize} \quad \sum_{(i,f)\in \boldsymbol{K}_t \setminus \boldsymbol{J}_t} w_{if} x_{ifkt} \tag{16}$$

$$\text{subject to} \quad x_{ifkt} \le y_{fkt}, \ (i,f) \in \boldsymbol{K}_t, \tag{17}$$

$$\sum_{f\in \boldsymbol{F}} s_f y_{fkt} + \sum_{(i,f)\in \boldsymbol{K}_t} p_{if} x_{ifkt} \le L, \tag{18}$$

$$x_{ifkt} \in \{0, 1\}, \ (i,f) \in \boldsymbol{K}_t, \tag{19}$$

$$y_{fkt} \in \{0, 1\}, \ f \in \boldsymbol{F}. \tag{20}$$

The order in which these problems are solved is based on the remaining available time, starting with the problem for the machine with the least available time in period *t*. In this way, the easiest jobs (with smaller processing times) are first tried to be added to the most loaded machines.

We now propose a number of options (j = 1, 2, …, 6) for the weight calculations $w_{if}^{(j)}$ of the knapsack problem of (16)-(20). The value of a weight is determined by two factors: a scaling factor γ_{if} and a preference indicator $v_{if}^{(j)}$:

$$w_{if}^{(j)} = \gamma_{if} v_{if}^{(j)} \tag{21}$$

The scaling factor γ_{if} gives preference to a job i belonging to family f of which there is already a job scheduled in period t on that machine. The factor is based on the average processing time $\overline{P} = \sum_{i=1}^{N} p_i / N$.

$$\gamma_{if} = \begin{cases} \dfrac{3\overline{P}}{2}, \text{with a family member on the same machine;} \\ \overline{P}, \text{with no family member;} \\ \dfrac{\overline{P}}{2}, \text{with a family member on another machine.} \end{cases}$$

For the preference indicator, we consider four basic options ($j \in \{1, 2, 3, 4\}$) and in addition two combinations of these ($j \in \{5, 6\}$). As already indicated, a more interesting job might be a job from a family with a large set-up time and especially when its family only contains a few yet to be processed jobs. In addition, care must be taken to process the job not too early.

$$v_{if}^{(1)} = \frac{TL}{d_{if} - D_t};$$

$$v_{if}^{(2)} = \frac{n_{\max}}{n'_f}, \text{ with}$$

$$\begin{cases} n_{\max} \text{ the number of jobs in the largest family} \\ \text{and } n'_f \text{ the number of jobs of family } f \text{ not yet scheduled;} \end{cases}$$

$$v_{if}^{(3)} = \frac{s_f + p_{if}}{p_{if}}.$$

A fourth preference indicator is proposed which aims to give preference to jobs with a due date equal to the end of more 'loaded' future periods. The initial load P_s^0 of a period $s \in \boldsymbol{T}$ can be defined as:

$$P_s^0 = \sum_{f\in F_s^0} s_f + \sum_{(i,f)\in S_s^0} p_{if}, \tag{22}$$

where $S_s^0 = \{(i,f) \in \boldsymbol{N} : d_{if} = sL\}$ and $F_s^0 = \{f \in \boldsymbol{F} : (i,f) \in S_s^0\}$. However, at the start of iteration $t \in \boldsymbol{T}$ some of these jobs may already have been allocated to previous periods. Therefore, define the subset S_s^t of jobs yet to be scheduled at iteration $t \in \boldsymbol{T}$ and with a due date at the end of period $s \in \boldsymbol{T}$, $s > t$:

$$S_s^t = \{(i,f) \in \boldsymbol{N} \cap S_t : d_{if} = sL\}.$$

Define a proper partition of the set S_s^t into a subset

$$S_s^{t,K} = \{(i,f) \in S_s^t : r_{if} < tL\} = S_s^0 \cap (\boldsymbol{K}_t \setminus \boldsymbol{J}_t)$$

of jobs having their release date before or at the start of period *t*, and another subset $S_s^{t,\varnothing}$ of jobs of which the release dates fall at the end or later than period *t*.

The jobs in $S_s^{t,K}$ are reordered according to non-increasing processing times. Let $S_s^{t,K}(i,f) \subseteq S_s^{t,K}$ denote the subset containing job (i, f) and all jobs of smaller order. For every period $s \in \boldsymbol{T}$, $s > t$ and every job (i, f) $\in S_s^{t,K}$, we now calculate a fourth preference indicator as follows:

$$v_{if}^{(4)} = \frac{P_s^{if}}{d_{if} - D_t},$$

where P_s^{if} is the sum of the processing times of the jobs that still have to be considered (in decreasing order) in period s and the set-up times of the corresponding families. So,

$$P_s^{if} = \sum_{f\in F_s^t(i,f)} s_f + \sum_{(i,f)\in S_s^{t,K}(i,f)\cup S_s^{t,\varnothing}} p_{if}$$

and

$$F_s^t(i,f) = \{f \in F : (i,f) \in S_s^{t,K}(i,f) \cup S_s^{t,\varnothing}\}.$$

The factor $v_{if}^{(4)}$ will make a job with a due date equal to the end of a more loaded period s more attractive for inclusion in the current period *t*, especially when this period is near to the current period. Furthermore, the proposed approach aims to relieve overloaded periods in a greedy fashion by giving jobs with larger processing times a larger weight.

In conclusion, the first preference indicator gives a larger weight to jobs of the nearer periods; it is calculated by considering the total number of periods divided by the number of periods that the job will be early. The second option is larger when there are only a few jobs not yet scheduled in the family compared to the maximum number of jobs in a family. The third expression favours a job with a large set-up time relative to its processing time. The last factor gives preference to jobs from more loaded periods: the load P_s^{if} is divided by the time the job will be early, when it is processed in the current period t.

A number of combinations of preference indicators are also considered:

$$v_{if}^{(5)} = v_{if}^{(1)} \times v_{if}^{(2)} \times v_{if}^{(3)}$$
$$v_{if}^{(6)} = v_{if}^{(1)} \times v_{if}^{(2)} \times v_{if}^{(3)} \times v_{if}^{(4)}.$$

In this way, distinct preferences can be given to two jobs of the same period but belonging to a small family or a family with a large set-up time ($v_{if}^{(5)}$). In addition, $v_{if}^{(6)}$ assigns a larger preference to the larger job of an overloaded period. Note that $v_{if}^{(2)}$, $v_{if}^{(3)}$ or $v_{if}^{(4)}$ are only calculated when some specific conditions are satisfied. When this is not the case, these factors are replaced by a value of 1 in the above formulas.

Thus, the heuristic approach proceeds iteratively for each value of t (t = 1, 2, …, T). First, the LPT rule is applied for allocating the jobs that have to be scheduled before the end of the period, to machines. Then, the knapsack-like problem (16)-(20) is solved for each machine with the same technique as we used for the single machine problem and all the appropriate job allocations to the current period are made. In (16)-(20) $w_{if}^{(j)}$ coefficients are precalculated by selecting one of the six presented options. Obviously, this approach can be easily adapted to a rolling horizon decision environment.

5. COMPUTATIONAL EXPERIENCE

In the previous section the heuristic approach presented by Crauwels and Van Oudheusden(2003) for the single machine case is extended to the parallel machine environment. In order to make computational comparisons between the two cases the test sets with T = 13 periods and a bucketlength L = 40 that were generated for the one machine case (Crauwels and Van Oudheusden, 2003), are used again. For the number of jobs N and the number of families F, we have used the combinations (N, F) = (20, 4); (30, 6) and (30, 10) to generate three test sets of 120 instances each.

Each test set is composed of four groups of thirty problems. The first group has an equally distributed load across the time horizon, where load is defined as in (22). For the second group this load varies periodically. The third and fourth group have an increasing and a decreasing load, respectively. According to this load a due date is assigned to each job. The release date is set to the corresponding due date minus a random number of periods. Each group contains three subsets of ten problems (five problems with uniformly distributed jobs across families and five with unequally distributed jobs).

The subsets have respectively a smaller, equal and larger average set-up time compared to the average processing time. These set-up and processing times are randomly generated integers from the uniform distribution defined on [1, B]. For equal ranges of set-up and processing times, the maximum B value, denoted $B_{\max}$, is equal to $\varrho TL/N$ with ϱ equal to the average load across the time horizon. For the subset with smaller set-up times, we use a maximum value of $2B_{\max}/3$ for the set-up times and $4B_{\max}/3$ for the processing times. For the subset with larger set-up times, this value is $4B_{\max}/3$ for the set-up times and $2B_{\max}/3$ for the processing times.

In order to get a comparable workload when several machines are available, the release dates and the due dates are adjusted:

$$r_j' = L \times \left\lfloor \frac{r_j}{ML} \right\rfloor \quad \text{and} \quad d_j' = L \times \left\lceil \frac{d_j}{ML} \right\rceil$$

for j = 1, …, N.

This also reduces the number of periods:

$$T' = \left\lceil \frac{\max_j d_j}{ML} \right\rceil. \tag{23}$$

By this reduction of the time horizon, it is possible that several jobs of a family have the same due date. In a single machine environment, this is not realistic because these jobs would be integrated into one large job so that only one set-up time has to be included. However, in a parallel machine environment it is possible that these jobs are scheduled on different machines in order to not overload the period.

The integer programming models described in Section 3 are solved by the 'gnu lp kit' of Makhorin (2006) on a HP 9000/rp5430. The performance is characterised by the average computation time in seconds (ACT), the number of unsolved problems (NU) because of a CPU time limit of 300 seconds and the average number of branch-and-bound nodes (ANN). When there are unsolved problems, the values listed under ACT and ANN are lower bounds on the true averages.

Table 1 presents the cumulated values of number of overloaded periods for each problem set with 120 instances when M = 1, 2, 3 machines are available. The row labeled with "initial" gives this cumulated value when each job is scheduled in the period just before its due date. For the row labeled with "optimal" the value is obtained by solving the first integer programming model with no limit on the possible overload in a period. By comparing the two rows corresponding to a test case, we see that a large improvement can be obtained by grouping together jobs of different periods whenever there is the opportunity.

In Table 2 the results on the number of overloaded periods model with a limit on overtime equal to L of the problem sets with N = 20 and 30 jobs are compared for one machine and two and three identical parallel machines.

Table 1. Cumulated number of overloaded periods

		N=20 F=4			N = 30 ϱ = 0.8	
		ϱ=0.7	ϱ=0.8	ϱ=0.9	F = 6	F = 10
M = 1	initial	357	443	546	514	441
	optimal	51	83	166	58	81
M = 2	initial	192	251	360	233	255
	optimal	38	66	154	41	63
M = 3	initial	119	176	270	152	169
	optimal	23	54	131	25	50

Most problems can be solved to optimality when there is only one machine. The problem with parallel machines is harder to solve and a lot of problem instances cannot be solved to optimality within the computation time limit of 300 seconds. When there are more parallel machines available, the problem gets harder. The effect of the other problem parameters is comparable with the one machine case. The problem set with a high load (ϱ = 0.9) is the most difficult to solve. Instances with more jobs are also more difficult to solve. The number of families parameter seems to have less effect on the performance of the IP model.

The results of the total overtime model (Table 3) show that these problems are still harder to solve. A lot of instances remain unsolved after 300 seconds of computation time. Again, the harder problems correspond with a higher average load, more jobs, and more machines. The average computation time for the test sets with N = 30 jobs on 2 and 3 machines is smaller than some ACT values for test sets with N = 20 jobs. This can be explained by the fact that for these test sets with N = 20 jobs a larger number of instances remains unsolved and each such instance 'consumes' the full 300 CPU seconds.

A conclusion of these first computational experiments is that the integer programming models, minimising the number of overloaded periods and minimising the total overtime, for the parallel machine problem are too hard to solve for realistic problem sizes.

Secondly, computational experiments were carried out following the heuristic approach described in Section 4, to examine the effectiveness of the different weights. Table 4 shows the results of the heuristic method regarding the number of overloaded periods. It tabulates the number of times the optimal solution (or, when the problem was not solved because of the computation time limit, the best known value) is found for each test set of 120 problems with N = 20 jobs and each type of weight.

For the non-combined weights, we observe that by using the first or fourth weight, an additional number of instances can be solved to optimality compared to the case where all weights are equal to one and no preference is giving to specific jobs. As for the single machine environment (see Crauwels and Van Oudheusden, 2003), weight $w_{if}^{(1)}$ performs quite well: it gives preference to jobs of nearer periods and by doing so, it can reduce the number of overloaded periods drastically. This is an interesting result because using weight $w_{if}^{(1)}$ in the knapsack problem can be translated into a simple rule for the dispatcher in the workshop: 'if there is some idle time in the current period on some machine, just add one or more jobs from the next period to the current period'.
An additional advantage of this rule is its applicability in a dynamic environment where the MRP software frequently reschedules planned orders. This rule only uses information of jobs from periods in the near future to construct the batches for the current period and changes in later periods have no impact at all. Weights $w_{if}^{(2)}$ and $w_{if}^{(3)}$, when used separately, do not improve on the number of times the optimal solution is found. However, when used in combination with the two other weights, a small improvement can be observed for some test sets. For a number of test sets, the best performance is obtained with weight $w_{if}^{(6)}$.

Analogous to the performance of the integer programming model, the heuristic method performs worse when the average load ϱ or the number of jobs increase. The number of parallel machines seems to have the opposite effect. When there are more machines available, more times the optimal value can be obtained and there is less difference between the different weights.

Apart from the number of overloaded periods other performance characteristics can be considered, e.g. the total overtime, the number of set-ups and the total earliness. Because optimal values for these objective values cannot be computed easily, the best known solution that is obtained from any of the methods using different weights, is recorded and the comparison is based on the number of times this best known solution is found. Tables 5, 6 and 7 show the results for the test sets with N = 30 jobs. Table 5, which shows the performance on the number of overloaded periods, confirms our findings of Table 4. Weight $w_{if}^{(1)}$ giving preference to jobs of nearer periods, performs quite well. The more complex weights $w_{if}^{(4)}$, $w_{if}^{(5)}$ and $w_{if}^{(6)}$ give analogous results. For example, the number of instances where the deviation from the best known solution is more than one overloaded period turns out to be 21 (on a total of 720 problems) with weight $w_{if}^{(1)}$, compared to 10 instances when weight $w_{if}^{(4)}$ is used. This weight $w_{if}^{(4)}$ was specifically defined for obtaining a minimal number of overloaded periods, because it gives preference to the jobs of the more loaded periods. Although weight $w_{if}^{(2)}$ and weight $w_{if}^{(3)}$ perform a little worse than the other weights for the number of times the best solution is found, the deviation from this best known solution is in most cases only one extra overloaded period.

The computational results for the total overtime performance characteristic are quite similar to the ones for the number of overloaded periods characteristic. The best results are almost always obtained with the combined weight $w_{if}^{(6)}$. And also weight $w_{if}^{(1)}$ performs quite well.

Table 2. Performance of integer programming model: number of overloaded periods

			one machine			two machines			three machines		
N	F	ϱ	ACT	NU	ANN	ACT	NU	ANN	ACT	NU	ANN
20	4	0.7	4.9	1	11168	16.1	3	33027	46.9	14	98069
		0.8	9.0	0	16687	52.6	14	102574	80.2	24	153879
		0.9	31.2	5	58297	110.2	37	185562	172.4	63	282756
30	6	0.8	38.2	13	77747	62.4	21	117412	51.1	19	93707
	10	0.8	26.8	4	50848	49.0	15	95524	65.3	21	132132

Table 3. Performance of integer programming model: total overtime

			one machine			two machines			three machines		
N	F	ϱ	ACT	NU	ANN	ACT	NU	ANN	ACT	NU	ANN
20	4	0.7	9.7	1	18765	21.6	5	38943	54.4	20	109684
		0.8	9.6	2	16806	59.4	16	105921	106.3	39	170279
		0.9	17.8	4	28678	122.8	39	190870	182.7	78	239566
30	6	0.8	34.6	9	57510	78.1	28	110579	58.9	23	83099
	10	0.8	36.8	10	58848	76.2	24	115408	96.0	37	159017

Table 4. Performance characteristic: number of overloaded periods

M	N	F	ϱ	$w_{if}=1$	$w_{if}^{(1)}$	$w_{if}^{(2)}$	$w_{if}^{(3)}$	$w_{if}^{(4)}$	$w_{if}^{(5)}$	$w_{if}^{(6)}$
1	20	4	0.7	80	98	85	85	105	99	105
			0.8	48	78	58	57	77	77	80
			0.9	45	62	45	45	68	58	63
2	20	4	0.7	85	93	91	90	94	94	94
			0.8	63	77	62	60	80	76	79
			0.9	35	42	37	37	40	42	41
3	20	4	0.7	101	104	101	102	104	104	104
			0.8	75	81	81	81	81	83	82
			0.9	51	49	49	50	54	50	55

Table 5. Performance characteristic: number of overloaded periods

M	N	F	$w_{if}=1$	$w_{if}^{(1)}$	$w_{if}^{(2)}$	$w_{if}^{(3)}$	$w_{if}^{(4)}$	$w_{if}^{(5)}$	$w_{if}^{(6)}$
1	30	6	67	103	76	76	102	101	105
	30	10	75	99	84	84	95	103	104
2	30	6	94	112	111	110	112	111	118
	30	10	93	113	105	104	114	115	114
3	30	6	110	119	113	113	117	118	119
	30	10	107	116	111	111	118	115	119

Table 6. Performance characteristic: total number of set-ups

M	N	F	$w_{if}=1$	$w_{if}^{(1)}$	$w_{if}^{(2)}$	$w_{if}^{(3)}$	$w_{if}^{(4)}$	$w_{if}^{(5)}$	$w_{if}^{(6)}$
1	30	6	48	43	60	54	49	52	44
	30	10	49	44	63	59	53	57	45
2	30	6	49	52	78	72	67	65	69
	30	10	50	62	79	80	70	73	79
3	30	6	48	66	74	68	68	76	78
	30	10	52	79	83	79	80	84	87

Table 7. Performance characteristic: total number of periods too early

M	N	F	$w_{if}=1$	$w_{if}^{(1)}$	$w_{if}^{(2)}$	$w_{if}^{(3)}$	$w_{if}^{(4)}$	$w_{if}^{(5)}$	$w_{if}^{(6)}$
1	30	6	44	33	22	22	42	26	46
	30	10	42	39	24	25	54	29	58
2	30	6	86	55	57	56	58	50	53
	30	10	93	64	68	69	72	63	68
3	30	6	92	67	60	64	61	65	58
	30	10	96	75	67	68	75	72	70

The computational requirement for the heuristic method is very low, less than one second for a problem instance. Although a lot of knapsack problems (16)-(20) have to be solved for each instance, the number of candidate jobs (elements of $\boldsymbol{K}_t$) is quite small. For the problem instances with 30 jobs, the number is on average equal to six with a maximum of 17 candidate jobs for some single machine problem instances.

When the primary objective is to save many set-ups, weight $w_{ij}^{(2)}$ should be used according to Table 6. This weight gives preference to jobs of rare families: when a job of a family is sequenced in the current period and there is just one other non-scheduled job in this family, it is best to include that job in the current period and on the same machine, if possible. The maximum deviation from the best known solution is for all weights on average five additional set-ups. Only the results with equal weights and with weight $w_{ij}^{(1)}$ show a slightly larger maximum deviation.

Table 7 presents the results for the total earliness performance measure. In the context of traditional lotsizing, total earliness correlates with the inventory cost. For the single machine case, weight $w_{ij}^{(6)}$ gives the best results. Weight $w_{ij}^{(4)}$ performs more or less similarly. With these two weights, jobs from more loaded periods get preference and as a result, less jobs are rescheduled from their due date period to earlier periods. With the other weights, it is possible that jobs from the next period are shifted into the actual period, thereby creating additional available time in that next period. As a consequence, additional jobs from subsequent periods are shifted. In this way, a lot of jobs can be assigned to earlier periods.

The maximum deviation from the best known solution is for the single machine case with equal weights and weights $w_{ij}^{(2)}$ and $w_{ij}^{(3)}$ quite large, more than twenty periods. For the other weights, this deviation is somewhat smaller, but still more than 15. For the parallel machine cases, this maximum deviation is smaller than ten for all weights and the smallest maximum deviation is observed for the equal weights case.

Rather remarkable is that in the multi-machine environment, the case where all weights are equal to one, performs best. With equal weights, the knapsack algorithm gives preference to the smaller jobs. In the first periods, it is possible that there is only capacity left for adding just one job from a following period. With equal weights, a small job is chosen and, as a consequence, in the following period also, not that much additional capacity becomes available. When here again a small job is added, we get the same phenomenon. As a result, only a few jobs will be shifted from subsequent to earlier periods resulting in a less total number of periods too early. When non-equal weights are used, larger jobs are shifted to previous periods, thereby creating more additional capacity in a following period, so that possibly more than one job can be shifted into that period. This is confirmed by the fact that with equal weights the average idle time in each period is larger that what is observed with non-equal weights.

The results presented for the test sets with N = 20 and 30 jobs seem to indicate that the difference between the weights is less pronounced when there are more parallel machines. But in these test sets the number of periods is reduced according to the number of machines in order to get a comparable load across the time horizon (see (23)). For example, a test set used in combination with M = 3 machines only considers T' = 5 periods compared to the T = 13 periods in the single machine environment. Maybe, because of this small number of periods, there is less opportunity for constructing a better solution based on some particular weight.

Therefore, new test sets with N = 60 jobs, F = 5, 10, 15 families and T = 13 periods are generated. In order to get a comparable workload when a different number of machines is used, the set-up and processing times are randomly generated integers from the uniform distribution defined on [1, B] with the maximum B value also depending on the number of machines: $B_{\max} = \varrho MTL/N$ with $\varrho = 0.9$.

Tables 8, 9 and 10 show the results for these new test sets with M = 2, 3 and 4 machines. In these tables, we have replaced the results for weights $w_{ij}^{(2)}$ and $w_{ij}^{(3)}$ by the results for the preference indicators $v_{ij}^{(1)}$ and $v_{ij}^{(4)}$ in order to show the effect of the scaling factor γ_{ij}. These two weights $w_{ij}^{(2)}$ and $w_{ij}^{(3)}$ only perform good for the total number of set-ups characteristic, as is shown in the tables with N = 30 jobs.

In most cases, our findings from the test sets with N = 30 jobs are confirmed. For the performance characteristics 'number of overloaded periods' (Table 8) and 'total number of set-ups' (Table 9), the performance can be improved by using non-equal weights. The difference between the weights becomes more pronounced when there are a lot of small families (F = 15). For the sets with only a few families, a large part of the instances can be solved to a feasible solution with no overloaded periods irrespective of the kind of weight that is used. For example, in the set with F = 5 families only 9 instances have an overloaded period when 4 machines are considered. For the number of overloaded periods (Table 8), weight $w_{ij}^{(1)}$ again performs quite well, and there is no reason for using the more complex weight $w_{ij}^{(5)}$. For a number of cases, weights $w_{ij}^{(4)}$ and $w_{ij}^{(6)}$ give slightly better results.

Tables 8 and 9 show that there is a positive effect on the results by using the scaled weights $w_{ij}^{(1)}$ instead of just the preference indicators $v_{ij}^{(j)}$. The effect is more pronounced for $w_{ij}^{(4)}$ than for $w_{ij}^{(1)}$.

For the performance characteristic 'total number of periods too early' (Table 10), the best performance is obtained by using equal weights in most cases. It is remarkable that the preference indicator $v_{ij}^{(4)}$ also performs quite well, a behaviour that is also observed in the single machine problem.

Table 8. Performance characteristic: number of overloaded periods

M	N	F	$w_{ij}=1$	$v_{ij}^{(1)}$	$v_{ij}^{(4)}$	$w_{ij}^{(1)}$	$w_{ij}^{(4)}$	$w_{ij}^{(5)}$	$w_{ij}^{(6)}$
2	60	5	98	113	99	118	118	117	118
	60	10	83	108	83	114	113	110	115
	60	15	60	91	66	96	102	95	112
3	60	5	98	116	104	117	115	114	117
	60	10	78	109	94	112	110	111	114
	60	15	61	101	75	103	105	101	106
4	60	5	114	119	117	120	118	119	118
	60	10	104	115	113	116	118	116	118
	60	15	79	111	94	114	111	114	115

Table 9. Performance characteristic: total number of set-ups

M	N	F	$w_{ij}=1$	$w_{ij}^{(1)}$	$w_{ij}^{(2)}$	$w_{ij}^{(3)}$	$w_{ij}^{(4)}$	$w_{ij}^{(5)}$	$w_{ij}^{(6)}$
2	60	5	16	1	20	3	30	10	8
	60	10	19	3	18	6	25	18	7
	60	15	21	6	21	7	27	16	22
3	60	5	15	5	18	11	21	13	18
	60	10	18	2	21	9	25	12	16
	60	15	32	7	21	7	27	12	27
4	60	5	12	4	12	15	24	18	20
	60	10	11	6	16	14	32	23	26
	60	15	20	15	29	14	38	20	33

Table 10. Performance characteristic: total number of periods too early

M	N	F	$w_{ij}=1$	$w_{ij}^{(1)}$	$w_{ij}^{(2)}$	$w_{ij}^{(3)}$	$w_{ij}^{(4)}$	$w_{ij}^{(5)}$	$w_{ij}^{(6)}$
2	60	5	56	18	34	4	7	2	15
	60	10	35	23	41	8	6	3	12
	60	15	37	19	36	9	16	7	19
3	60	5	72	14	46	6	5	6	5
	60	10	54	16	52	4	10	2	13
	60	15	48	15	60	4	5	2	7
4	60	5	61	10	60	5	5	4	3
	60	10	60	16	48	6	4	5	3
	60	15	63	10	43	5	7	6	13

6. CONCLUSIONS

This paper presents some methods for scheduling planned MRP orders consisting of jobs with family set-up times in a parallel machine environment. The computational experiments on the integer programming models illustrate the combinatorial nature of the problem. A number of problems with only twenty jobs cannot be solved to optimality within a moderate computation time. Furthermore, with the first model it is not always possible to construct a feasible schedule because, for some problem instances, more overtime is required in some periods than is allowed by the model.

The proposed heuristic approach does not suffer from these two drawbacks. A detailed job sequence can be computed simply and quickly. The approach also aims at satisfying a number of performance criteria, such as reducing the overload and set-ups without heavily increasing the earliness of some jobs. Equal use of the capacity in each period, especially in the beginning of the time horizon, is also obtained. The simple rule where jobs of the nearer periods are selected for inclusion in the current period, performs very well on a large number of important criteria. By changing the nature of the weights in the knapsack problem formulation, the heuristic algorithm can furthermore compute sequences that perform well on very specific criteria.

The proposed heuristic approach, using a knapsack-like problem with different possible weights to 'optimise' the available resources of the current time period, seems to be a practical and very flexible decision device that, very likely, can be useful in different short-term scheduling environments. The study and computational experiments in this paper clearly indicate the applicability and usefulness of this method in the realistic context of MRP and parallel machine scheduling, with families of jobs sharing a common set-up time. Thus, in many cases, the proposed approach can be a valuable alternative to dynamic, capacitated lot-sizing techniques often recommended in the MRP context. It is indeed easy to incorporate the typical lot-sizing trade-off between inventory and set-up

cost by means of careful weight selection.

The approach can be extended in several ways. Firstly, the constraint that each period of the time horizon has to be of equal length, is easily relaxed. Secondly, it is possible that in some processing environments the first set-up in a time period is not necessary when the first job of the period belongs to the same family as the last job of the previous period. Another possible complication is preemption: when a job cannot be completely processed in one period, it can without any set-up be continued in the next period. An area that should be further investigated is when costs are not proportional to time. Probably, the current set of weights can be extended to incorporate cost related trade-offs. Also interesting for future research are sequence-dependent set-up times and costs both in a single and parallel machine environment.

REFERENCES

1. Allahverdi, A., Gupta, J.N.D., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, *International Journal of Management Science*, 27: 219-239.
2. Allahverdi, A., Ng, C.T., Cheng, T.C.E., and Kovalyov, M.Y. (2006). A survey of scheduling problems with setup times or costs. *European Journal of Operations Research* (to appear).
3. Baker, T. and Muckstadt, J.A., Jr. (1989). *The CHESS problems*. Technical Paper, Chesapeake Decision Sciences, Inc., 200 South Street, New Providence, NJ.
4. Clark, A. (2003). Approximations for capacity constrained MPS problems. *International Journal of Production Economics*, 84: 115-131(internal research report MS-2002-2, School of Mathematical Sciences, University of the Westof England, Bristol).
5. Chen, Z.-L. and Powell, W.B. (2003). Exact algorithms for scheduling multiple families of jobs on parallel machines. *Naval Research Logistics*, 50 (7): 623-640.
6. Crauwels, H.A.J. and Van Oudheusden, D. (2003). Transforming an MRP plan into a short-term schedule. *Production Planning & Control*, 14 (7): 647-655.
7. Drexl, A. and Kimms, A. (1997). Lot sizing and scheduling: survey and extensions. *European Journal of Operational Research*, 99: 221-235.
8. Dunstall, S. and Wirth, A. (2005). Heuristic methods for the identical parallel machine flowtime problem with set-up times. *Computers & Operations Research*, 32: 2479-2491.
9. Kang, S., Malik K., and Thomas, L.J. (1999). Lotsizing and scheduling on parallel machines with sequence-dependent setup costs. *Management Science*, 45 (2): 273-289.
10. Maes, J. and Van Wassenhove, L.N. (1988). Multi-item single-level capacitated dynamic lot-sizing heuristics: a general review. *Journal of the Operational Research Society*, 39(11): 991-1004.
11. Magnusson, T. (2001). *The capacitated lot-sizing and scheduling problem with sequence-dependent setup costs and setup times*. Master of Science thesis, Graduate School of the University of Minnesota.
12. Makhorin, A. (2006). *GLPK* (*GNU Linear Programming Kit*), Department for Applied Informatics, Moscow Aviation Institute, Russia. Available: http://www.gnu.org/software/glpk/glpk.html.
13. Martello, S. and Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. Johm Wiley & Sons Ltd.
14. Meyr, H. (2002). Simultaneous lotsizing and scheduling on parallel machines. *European Journal of Operational Research*, 139: 277-292.
15. Pisinger, D. (1999). An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114: 528-541.
16. Potts, C.N. and Kovalyov, M.Y. (2000). Scheduling with batching: a review. *European Journal of Operational Research*, 120: 228-249.
17. Potts, C.N. and VanWassenhove, L.N. (1992). Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of the Operational Research Society*, 43: 395-406.
18. Quadt, D. and Kuhn, H. (2003). Production planning in semiconductor assembly. *Fourth Aegean International Conference on "Analysis of Manufacturing Systems"*, University of the Aegean, Samos Island, Greece, pp. 181-189.
19. Staggemeier, A.T. and Clark, A. (2001). *A survey of lot-sizing and scheduling models*. University of the West of England, Bristol.
20. Webster, S. and Baker, K.R. (1995). Scheduling groups of jobs on a single machine. *Operations Research*, 43: 692-703.