Multiplicity and complexity issues in contemporary production scheduling

N. Brauner[*], Y. Crama[†], A. Grigoriev[‡], J. van de Klundert[§]

Abstract

High multiplicity scheduling problems arise naturally in contemporary production settings where manufacturers combine economies of scale with high product variety. Despite their frequent occurrence in practice, the complexity of high multiplicity problems - as opposed to classical, single multiplicity problems - is in many cases not well understood. In this paper, we discuss various concepts and results that enable a better understanding of the nature and complexity of high multiplicity scheduling problems. The paper extends the framework presented in Brauner et al. (2005) for single machine, non preemptive high multiplicity scheduling problems, to more general classes of problems.

Keywords: computational complexity, design of algorithms, scheduling, high multiplicity

We dedicate this paper to the memory of Antoon Kolen. Antoon has founded and led the Operations Research Group at the Faculty of Economics and Business Administration of Maastricht University from 1985 till 2004. Throughout this period, he has stimulated the application of fundamental combinatorial methods to new and challenging problems arising in innovative settings. Since 1985, automated manufacturing has been one of the fields of interest of the group. The ideas presented in this paper find their origins in the research activities of this group.

[*]Laboratoire Leibniz-IMAG, 46 avenue Félix Viallet, 38031 Grenoble cedex, France, Nadia.Brauner@imag.fr

[†]HEC Management School, University of Liège, Boulevard du Rectorat 7 (B31), 4000 Liège, Belgium, Y.Crama@ulg.ac.be

[‡]Department of Quantitative Economics, Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands, A.Grigoriev@ke.unimaas.nl

[§]Department of Mathematics, Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands, J.vandeKlundert@math.unimaas.nl

1 Prelude

Manufacturing innovation, in particular the adoption of flexible manufacturing systems, has made it possible to produce small or medium quantities of various products on a same line or group of machines without incurring costly setups between batches of different product types. In such settings, it is common to cyclically repeat a production schedule in which the items are produced according to the ratios defined in the tactical plan. Hence, the input of the associated scheduling problem relies on the ratios in which jobs of the various types are to be produced, rather than on a description of each individual job. As a consequence, for many such problems the size of the traditional encoding of a solution, namely an explicit schedule or sequence, may not be polynomial in the input size of the problem. We illustrate this phenomenon on a couple of flowshop scheduling problems.

Consider a bufferless two-machine flowshop which processes jobs of s different types. For each $i = 1, \ldots, s$, there are n_i identical jobs of type i; each such job must successively undergo two operations O_{i1} and O_{i2}, to be executed on machines 1 and 2 respectively. The processing requirements of operation O_{ik} are summarized by its processing time p_{ik}, for $i = 1, \ldots, s$ and $k = 1, 2$; note that p_{ik} only depends on i and k, since all jobs of type i are identical. As usual, we also assume that no two jobs can simultaneously occupy a machine. The task is now to find a schedule, i.e. starting times for all individual operations, in which jobs of each type i are produced in quantity n_i, $i = 1, 2, \ldots, s$, and such that the output rate of the flowshop (in jobs per time unit) is maximized when the schedule is cyclically executed infinitely often. Note that the total number of jobs $n = n_1 + n_2 \ldots + n_s$ is not necessarily polynomial in s, and indeed not necessarily polynomial in the input size.

The single multiplicity version of this problem, where $n_1 = n_2 = \ldots = n_s = 1$, can be solved in $O(s \log s)$ time by a well-known algorithm for minimizing the makespan in two-machine flowshops; see Gilmore and Gomory (1964). A straightforward application of this algorithm to the high multiplicity case yields an algorithm of complexity $O(n \log n)$, which is not polynomial in the input size. Therefore, the question arises whether the high multiplicity version of the problem is polynomially solvable.

In order to obtain a more precise formulation of this question, let us note that the Gilmore-Gomory algorithm simply sorts the jobs in order to determine the optimal processing sequence. Applying the same sorting procedure to the high multiplicity version of the problem yields a sequence where jobs of a same type occur consecutively. Without going into the details yet, we mention that this property allows to compute a starting time *for any individual job* in polynomial time. On the other hand, *the length of the complete sequence* of n jobs (which constitutes a standard polynomial encoding of a solution for classical, single multiplicity scheduling problems) is superpolynomial in the input length. An intriguing question is therefore: what do we exactly mean by the requirement to "solve a high multiplicity scheduling problem"?

For another example, consider the three-machine flowshop problem. In comparison with the two-machine case, the input of the three-machine flowshop problem is obtained by adding the processing time of each operation on the third machine, say $p_{i3}, i = 1, \ldots, s$. The single multiplicity decision version of this problem is known to be NP-complete, see e.g. Garey, Johnson and Sethi (1976), and since it is a special case of the high multiplicity version, the latter cannot be polynomially solvable unless P=NP. On the other hand, when s is fixed (say, when $s = 2$), the single multiplicity version is trivially solved in constant time by enumeration. However, the same approach does not yield a polynomial time algorithm for the high multiplicity case with constant s. In fact, it is not trivial to establish that the problem is in NP, or in PSPACE, since encoding a solution as a list which contains an item for each job requires superpolynomial space. Generally speaking, it can be stated that the development of efficient algorithms for high multiplicity scheduling problems always depends on a deep understanding of the combinatorial nature of their solutions.

The goal of this paper is to addresses some of the issues raised by the above examples. In particular, it generalizes the framework proposed in Brauner et al. (2005) for the analysis of the complexity of high multiplicity scheduling problems, and it further advances the work on encoding schemes for their solution.

2 High multiplicity problems

Hochbaum and Shamir (1990, 1991) have introduced the term "high multiplicity" in combinatorial optimization, and have stressed the need to discuss the complexity of high multiplicity scheduling problems with special care. Clifford and Posner (2001) have pursued the topic and have described a more general setting for its complexity analysis. A thorough discussion, as well as a proposal for complexity classification of non-preemptive single machine scheduling problems, can be found in Brauner et al. (2005). The present paper generalizes and extends the work initiated by Brauner et al. To do so, we restate part of their definitions and framework in a multi-machine, preemptive context, and for multi-stage scheduling problems. We subsequently generalize most of the results found in Brauner et al. (2005) and we apply the generalized results to multiple machine applications.

The input of a classical scheduling problem $\mathcal{SP}$ consists of a list of n jobs, together with a list of attributes of each job. The attributes of job j ($j = 1, 2, \ldots, n$) typically include its processing time p_{jk} on each machine $k, k = 1, \ldots, m$, its release date r_j, its due date d_j, etc. The binary input size of an instance of $\mathcal{SP}$ is $O(n \times L)$, where L is the largest input size of an attribute.

In contrast, the input of a high multiplicity scheduling problem $\mathcal{SP}$ consists of the following data:

– the number of job types, viz. s;

– for each job type $i = 1, 2, \ldots, s$, the number of jobs of type i, viz. n_i ;

– for each job type $i = 1, 2, \ldots, s$, the attributes of a representative job of type i.

So, a generic instance of a high multiplicity scheduling problem $\mathcal{SP}$ takes the form $D = (s, n_1, n_2, \ldots, n_s, \Delta)$, where Δ comprises all the relevant job attributes, such as the processing times on the machines. We assume without loss of generality that all the entries of D are integral, and that the jobs are numbered from 1 to n in such a way that jobs 1 to n_1 are of type 1, jobs $n_1 + 1$ to $n_1 + n_2$ are of type 2, etc.

If we denote by $|D|$ the input size of an instance D of a high multiplicity scheduling problem, then $|D| = O(\sum_{1 \leq i \leq s} \log n_i + sL) = O(s \log n + sL)$, where L is again the largest input size of an attribute value and $n = \sum_{1 \leq i \leq s} n_i$. Typically, this input size is much smaller than nL, as is e.g. the case when s is viewed as a constant (as in the 3-machine flowshop problem with 2 job types described in the prelude). More precisely, we say that $\mathcal{SP}$ is a high multiplicity scheduling problem if n is not polynomially bounded in the input size of the problem, i.e. if there is no constant k such that $n = O\left((sL)^k\right)$ for all instances of $\mathcal{SP}$. Thus, an algorithm for $\mathcal{SP}$ whose complexity is polynomial in s, L *and* n is only *pseudo-polynomial*, but not polynomial in the input size.

3 Schedules and problem formulations

To start the analysis, we first give a formal definition of a *schedule*.

Definition 1 *For an instance involving n jobs and m machines, we define a* schedule *to be a (finite) subset S of $\{1, 2, \ldots, n\} \times \{1, 2, \ldots, m\} \times \mathbb{R} \times \mathbb{R}$. We use the generic notation: $q = |S|$.*

The interpretation of a schedule is that, if $(j, k, t_1, t_2) \in S$, then job j must be processed on machine k without preemption from time t_1 to time t_2. Note that this definition allows for (finitely many) preemptions. On the other hand, we assume as usual that every machine can only process one job at a time, and that every job requires only one type of operation on each machine (we do not consider reentrant flows).

Let us now turn to the objective function of $\mathcal{SP}$. Let $\mathcal{F}_D$ denote the set of feasible schedules associated with the instance D, and let $f_D : \mathcal{F}_D \to \mathbb{R}$ be the *objective function* to be minimized over $\mathcal{F}_D$. For instance, $f_D(S)$ could measure the makespan or the weighted tardiness of the schedule S. For the sake of simplicity, we assume that $\mathcal{F}_D$ is non empty for every instance D, and that f_D always attains its minimum over $\mathcal{F}_D$.

For all practical purposes, we henceforth assume that, given a description of S in extension (i.e., given a list of the elements of S), $f_D(S)$ can be computed in time polynomial in $|D|$ and q (this is a rather weak assumption).

Now that the solution space and the objective function of scheduling problems are properly defined, we introduce three distinct scheduling problems associated with $\mathcal{F}_D$ and f_D, in the spirit of Brauner et al. (2005) and Papadimitriou and Steiglitz (1982):

RECOGNITION PROBLEM $\mathcal{SP}_1$:
INSTANCE: $D = (s, n_1, n_2, \ldots, n_s, \Delta)$ and $K \in \mathbb{R}$.
OUTPUT: Yes if there is a schedule $S \in \mathcal{F}_D$ with $f_D(S) \leq K$. No otherwise.

EVALUATION PROBLEM $\mathcal{SP}_2$:
INSTANCE: $D = (s, n_1, n_2, \ldots, n_s, \Delta)$.
OUTPUT: The minimum value of f_D over $\mathcal{F}_D$.

OPTIMIZATION PROBLEM $\mathcal{SP}_3$:
INSTANCE: $D = (s, n_1, n_2, \ldots, n_s, \Delta)$.
OUTPUT: A schedule $S \in \mathcal{F}_D$ which minimizes $f_D(S)$ over $\mathcal{F}_D$.

Issues related to the complexity classification of $\mathcal{SP}_1$ or $\mathcal{SP}_2$ fall within the traditional scope of complexity analysis, as discussed, e.g., by Garey and Johnson (1979) or Papadimitriou and Steiglitz (1982). However, as already noted, since membership in NP (or in co-NP) is often non trivial to establish for high multiplicity scheduling problems, their complexity analysis can be much more cumbersome for such problems than for their single multiplicity counterparts. In fact, for several high multiplicity scheduling problems, it is open whether they are in NP (or in co-NP). On the other hand, many high multiplicity scheduling problems have been proved to be polynomially solvable (Agnetis (1997), Brauner, Finke and Kubiak (2003), Clifford and Posner (2000, 2001), Granot and Skorin-Kapov (1993), Hochbaum and Shamir (1990, 1991), Hochbaum, Shamir and Shanthikumar (1992), Hurink and Knust (2001), McCormick, Smallwood and Spieksma (2001)) or to be in co-NP (Brauner and Crama (2004)), or to be NP-hard (Bar-Noy et al. (2002), Clifford and Posner (2000, 2001), Posner (1985)). Such results and other similar results found in the literature can be established by displaying optimality or feasibility certificates whose size is polynomial in the input length $O(s \log n + sL)$. Such certificates, clearly, cannot provide a list of all the elements of S, but rather provide an implicit, concise encoding of S. All these issues are explicitly addressed in Example 3 in Section 5 which continues the analysis of the flowshop scheduling problem introduced in Section 1.

4 Solving the optimization version of high multiplicity scheduling problems

4.1 List-generating algorithms

This section considers the issue of solving the optimization version $\mathcal{SP}_3$ of a high multiplicity scheduling problem, where the output consists of a schedule as introduced in Definition 1. We first consider a framework to analyze the complexity of high multiplicity scheduling problems, as it has been presented in Brauner et al. (2005). This framework introduced several complexity classes, which appear to capture some of the peculiarities of high multiplicity problems well. It assumes that the set S is to be generated in extension:

Definition 2 *An algorithm $\mathcal{A}$ is a* list-generating algorithm *for problem $\mathcal{SP}_3$ if, for every instance of $\mathcal{SP}_3$, $\mathcal{A}$ successively outputs the elements $(\pi^1, \mu^1, t_1^1, t_2^1)$, $(\pi^2, \mu^2, t_1^2, t_2^2)$, $\ldots$, $(\pi^q, \mu^q, t_1^q, t_2^q)$ of an optimal schedule S, where $q = |S|$.*

For a list-generating algorithm $\mathcal{A}$, we let $\tau(0) = 0$ and for $h = 1, \ldots, q$, we denote by $\tau(h)$ the running time required by $\mathcal{A}$ in order to output the first h elements of the schedule, i.e. $(\pi^1, \mu^1, t_1^1, t_2^1), (\pi^2, \mu^2, t_1^2, t_2^2), \ldots, (\pi^h, \mu^h, t_1^h, t_2^h)$. So, $\tau(q)$ is the total running time of $\mathcal{A}$, and $\tau(h) - \tau(h-1)$ is the time elapsed between the $(h-1)$-st and the h-th outputs.

The classification of list-generating algorithms to be described below in Definition 3 is based on a proposal due to Johnson, Yannakakis and Papadimitriou (1988), for problems in which the size of the output may be exponentially larger than the size of the input such as, for instance, the problem of listing all maximal independent sets of a graph, or all vertices of a polyhedron;

see also Dyer (1983) or Lawler, Lenstra and Rinnooy Kan (1980) for related concepts. A similar approach is encountered in the study of the complexity of counting and enumerating solutions of multicriteria problems; see T'kindt, Bouibede-Hocine and Esswein (2005) for a thorough treatment.

Definition 3 *A list-generating algorithm $\mathcal{A}$ for $\mathcal{SP}_3$ runs:*

- *in* pseudo-polynomial time *if $\tau(q)$ is polynomially bounded in $|D|$ and M, where M is the largest number appearing in D;*
- *in* polynomial total time *if $\tau(q)$ is polynomially bounded in q and $|D|$;*
- *in* polynomial incremental time *if $\tau(h)-\tau(h-1)$ is polynomially bounded in h and $|D|$, for $h = 1, \ldots, q$;*
- *with* polynomial delay *if $\tau(h) - \tau(h-1)$ is polynomially bounded in $|D|$, for $h = 1, \ldots, q$;*
- *in* polynomial time *if $\tau(q)$ is polynomially bounded in $|D|$.*

Pseudo-polynomial time and polynomial time are the usual concepts from complexity theory and are only mentioned here for the sake of completeness. In particular, if there exists a polynomial time list-generating algorithm for $\mathcal{SP}_3$, then q and hence n are bounded by a polynomial in $|D|$ for all instances of this problem, and the problem does not qualify as a high multiplicity problem. On the other hand, if $\mathcal{SP}_3$ can be solved in pseudo-polynomial time, then the same complexity holds for $\mathcal{SP}_1$ and $\mathcal{SP}_2$ (since we assumed that $f_D(S)$ can be computed in time polynomial in $|D|$ and q).

Polynomial total time is, in a sense, the weakest notion of polynomiality which can be applied to $\mathcal{SP}_3$, since the running time of any algorithm which lists all elements of q must grow at least linearly with q.

Polynomial incremental time captures the idea that the algorithm outputs the elements of S sequentially and does not spend "too much time" between two successive outputs. In computing the elements however, the algorithm may need to look at the previous elements (for instance, to check feasibility of the partial schedule) and therefore we allow $\tau(h)-\tau(h-1)$ to depend on h as well as on $|D|$.

Finally, an algorithm runs with polynomial delay when the time elapsed between two successive outputs is polynomial in the input size of the problem. This is a rather strong requirement, the strongest, in fact, among those discussed in Johnson et al. (1988). We also feel that it is one of the most meaningful requirements that may apply to algorithms for high multiplicity scheduling problems.

For preemptive problems, the number of elements of an optimal schedule S is not easily determined as a function of the input parameters. However, for many classical preemptive scheduling problems there exists an optimal schedule S involving a number of preemptions per machine, which is polynomially bounded in the number of jobs n. For high multiplicity, this translates to a number of preemptions that is polynomial in $n \times m$. When this is the case, we require that $\mathcal{A}$ output a schedule S whose number of elements is polynomial in $n \times m$.

Proposition 1 *If $\mathcal{A}$ is a list-generating algorithm for the optimization version $\mathcal{SP}_3$ of a general scheduling problem, then:*

$$\begin{aligned} \mathcal{A} \textit{ runs in polynomial time} &\Longrightarrow \mathcal{A} \textit{ runs with polynomial delay} \\ &\Longrightarrow \mathcal{A} \textit{ runs in polynomial incremental time} \\ &\Longrightarrow \mathcal{A} \textit{ runs in polynomial total time.} \end{aligned}$$

Proof All the implications are easy. For instance, if $\mathcal{A}$ runs in polynomial incremental time, then the whole schedule can be generated in time $\tau(q) = \sum_{h=1}^{q} [\tau(h) - \tau(h-1)]$, which is polynomial in q and $|D|$. Hence, $\mathcal{A}$ runs in polynomial total time. □

Since the size of the optimal schedule q is not necessarily bounded by a polynomial in M, viz. the largest number occurring in the instance, the existence of a polynomial total time list-generating algorithm does not automatically imply the existence of a pseudo-polynomial list-generating algorithm. However, when q is polynomially bounded in $n \times m$ (cf. the discussion on preemption above), then polynomial total time can be seen to imply pseudo-polynomial time. Example 3 contains an application of this framework to the flowshop scheduling problem given in Section 1.

4.2 Pointwise algorithms

We now turn our attention to a different conceptual framework, where we assume that the aim of the solution procedure is no longer to generate *explicitly* all elements of the optimal schedule S, but only to be able to compute one of the mappings derived from S as explained below. The underlying idea is here that an *implicit* encoding of the schedule S (or an implicit encoding of the job sequence π) should suffice, if it can be decoded to produce useful information like the starting time of an arbitrary job or the state of a machine at any given time. To clarify this point, we first present several possible mappings associated to a schedule (see also Brauner et al. (2005) for the one-machine case).

Definition 4

- *The* machine-oriented *description of schedule S is the mapping*

$$S_M : k \mapsto S_M(k) = \{(j, t_1, t_2) \mid (j, k, t_1, t_2) \in S\}.$$

- *The* job-oriented *description of schedule S is the mapping*

$$S_J : j \mapsto S_J(j) = \{(k, t_1, t_2) \mid (j, k, t_1, t_2) \in S\}.$$

- *The* (machine, time)-oriented *description of schedule S is the mapping*

$$\begin{aligned} S_{MT} : \{1, 2, \ldots, m\} \times \mathbb{R} &\rightarrow \{0, 1, \ldots, n\} \times \mathbb{R} \times \mathbb{R} : \\ (k, t) \mapsto S_{MT}(k, t) &= (j, t_1, t_2) \quad \text{if } (j, k, t_1, t_2) \in S \text{ and either} \\ &\qquad t_1 \leq t \leq t_2 \text{ or } t_1 > t \\ &\qquad \text{and } t_1 \text{ is the first instant when} \\ &\qquad \text{a job is processed on machine } k \\ &\qquad \text{after time } t, \\ &= (0, 0, 0) \quad \text{if there are no more jobs to be} \\ &\qquad \text{processed on machine } k \text{ after} \\ &\qquad \text{time } t. \end{aligned}$$

- *For a nonpreemptive problem, the* (machine, sequence)-oriented *description of schedule S is the mapping*

$$\begin{aligned} S_{MS} : \{1, 2, \ldots, m\} \times \{1, \ldots, n\} &\rightarrow \{0, 1, \ldots, n\} : \\ (k, j) \mapsto S_{MS}(k, j) &= \pi_k(j) \quad \text{if job } j \text{ is processed} \\ &\qquad \text{on machine } k \text{ and job } \pi_k(j) \\ &\qquad \text{is its successor on machine } k \\ &= 0 \quad \text{if there are no more jobs} \\ &\qquad \text{to be processed} \\ &\qquad \text{on machine } k \text{ after job } j. \end{aligned}$$

Let us try to clarify these definitions and to explain the nature of the different descriptions. A machine-oriented description of the schedule gives the Gantt chart associated to each particular machine. A job-oriented description of the schedule describes the complete routing of a job through the shop. The mapping $S_{MT}(k, t)$ describes the state of machine k at time t. This (machine, time)-oriented description corresponds to the viewpoint of a human machine-operator, who must know, at every instant, which job is being processed or which job will be processed next on his machine. Such a human operator doesn't need to know the full explicit schedule, as long the current or next element of it is "explicitly" accessible. In some applications, the human operator doesn't even need to know when the jobs are processed: the only important information is the order in which the jobs must be processed. In this case, the (machine, sequence)-oriented mapping describes the successor of a job on a machine, where the successor of job j on machine k is defined as the job with minimum starting time on machine k among all jobs whose starting time on machine k exceeds the starting time of job j on machine k (this mapping is useful for non preemptive schedules).

A *pointwise (machine, time)-oriented algorithm* for $\mathcal{SP}_3$ is an algorithm which computes the mapping S_{MT} derived from an optimal schedule S, that is an algorithm which, on the input (D, k, t), outputs the value of $S_{MT}(k, t)$.

Similarly to Proposition 2 in Brauner et al. (2005), we can prove:

Proposition 2 *For an arbitrary scheduling problem* $\mathcal{SP}_3$
(a) *if* $\mathcal{SP}_3$ *has a polynomial list-generating algorithm, then* $\mathcal{SP}_3$ *has a polynomial pointwise (machine, time)-oriented algorithm;*
(b) *if* $\mathcal{SP}_3$ *has a polynomial pointwise (machine, time)-oriented algorithm, then* $\mathcal{SP}_3$ *has a polynomial-delay list-generating algorithm.*

Proof Assertion (a) holds trivially, since all the elements of an optimal schedule can be generated in polynomial time when a polynomial list-generating algorithm is available.

Conversely, if $\mathcal{A}$ is a polynomial pointwise (machine, time)-oriented algorithm, then, for each machine k, $\mathcal{A}$ can for instance be used to generate the jobs in order of their starting times as follows. Let $\mathcal{A}$ generate $S_{MT}(k, 0)$ in polynomial time. If it differs from $(0, 0, 0)$, this value $S_{MT}(k, 0) = (j, t_1, t_2)$ provides the index of the first job to be processed on machine k and its processing interval $[t_1, t_2]$. Subsequently compute $S_{MT}(k, t_2)$ in polynomial time, and so on, until $\mathcal{A}$ returns $(0, 0, 0)$. This sequence of steps solves $\mathcal{SP}_3$ with polynomial delay as required. $\square$

Hence, we conclude that polynomial pointwise (machine,time)-oriented algorithms fall somewhere between polynomial delay and polynomial time in the hierarchy presented in Proposition 1.

In contrast with Proposition 2 and with the results in Brauner et al. (2005), analyzing the complexity of the derived mappings S_J, S_M, or S_{MS} poses new difficulties, since mappings S_J and S_M are set-valued, rather than single-valued, and the mapping S_{MS} does not map to the time sets. The analysis can be carried out, however, by combining the notions of list-generating and pointwise algorithms. For instance, we could say that $\mathcal{A}$ runs (pointwise) with polynomial delay for S_J if, given any job j, $\mathcal{A}$ outputs the elements of the set $S_J(j)$ with polynomial delay. Such definitions may or may not prove useful or meaningful, depending on the context, and we will not dive deeper into their discussion.

5 Some high multiplicity problems revisited

Let us illustrate the concepts introduced in the previous sections on some examples of high multiplicity problems.

Clifford and Posner (2001) investigate the complexity of several parallel machine scheduling problems with high multiplicity. They establish that several of these problems are polynomially solvable both in their recognition and in their optimization versions (e.g., $P| \,|\sum_j C_j$ or $Q| \,|\sum_j C_j$), but they also argue that there is no polynomial description of the optimal schedule in terms of job groups for some other problems (e.g., $P|pmtn|C_{max}$ and $Q2|pmtn|\sum_j C_j$). We now show, however, that this does not preclude other efficient descriptions of the optimal schedule. We only handle two simple cases, as these suffice to illustrate our claim.

Example 1 ($P|pmtn|C_{max}$)

Consider first the makespan minimization problem on identical parallel machines with preemptions. An instance of the problem is a vector

$$D = (s, n_1, n_2, \dots, n_s, p_1, p_2, \dots, p_s, m),$$

where m is the number of machines and p_i is the processing time of a job of type i ($i = 1, 2, \dots, s$).

Clifford and Posner (2001) observe that the evaluation version of $P|pmtn|C_{max}$ can be solved in polynomial time. Indeed, in view of a well-known result of McNaughton (1959), the optimal value of this problem is equal to

$$C^*_{max} = \max\left\{\sum_{i=1}^{s} n_i p_i/m, p_1, p_2, \dots, p_s\right\},$$

which can be efficiently computed.

McNaughton's algorithm determines a schedule with makespan equal to C^*_{max}. It first lists all jobs in the natural order $1, 2, \ldots, n$. Then, it cuts this sequence, viewed as a single-machine schedule, into at most m subsequences of length C^*_{max}. Finally, the k-th subsequence is assigned to machine k, for $k = 1, 2, \ldots, m$; see McNaughton (1959) and Pinedo (1995).

Even with a single job type, the optimal schedule may require $\Omega(m)$ preemptions, where m is exponential in the input size. From this, Clifford and Posner (2001) conclude that "it is not possible to create an optimal schedule (...) in polynomial time" and hence, that the optimization version of $P|pmtn|C_{max}$ in in $EXP \backslash P$. This is rather surprising, in view of the simplicity of the evaluation problem $\mathcal{SP}_2$ and of McNaughton's algorithm. As a matter of fact, we note that the optimal schedule can actually be computed with polynomial delay. More precisely, the job-oriented application S_J can be computed (pointwise) with polynomial delay, as follows easily from the description of McNaughton's algorithm. This implies (as in Proposition 2) that an optimal schedule can be generated with polynomial delay. Similarly, the (machine,time)-oriented description S_{MT} can be computed pointwise in polynomial time. □

Example 2 ($Q2|pmtn|\sum_j C_j$)

An instance is a vector

$$D = (s, n_1, n_2, \ldots, n_s, p_1, p_2, \ldots, p_s, v_1, v_2),$$

where $p_1 < p_2 < \ldots < p_s$, v_1 (resp. v_2) denotes the speed of machine 1 (resp. machine 2) and $v_1 \geq v_2$. In an optimal schedule, the first job of type 1 starts on machine 1 at time 0. All other jobs start processing on machine 2 in SPT order, and are moved to machine 1 whenever this machine becomes available.

Clifford and Posner (2001) define the quantity $\sigma_i(j)$, representing the amount of time that the j-th job of type i spends on machine 1. They prove that, for $j = 1, 2, \ldots, n_i$ and $i = 1, 2, \ldots, s$,

$$\sigma_i(j) \;=\; \frac{p_i}{v_1 + v_2} - \left(\frac{p_i}{v_1 + v_2} - \sigma_i(0)\right)\left(-\frac{v_2}{v_1}\right)^j, \qquad (1)$$

where $\sigma_1(0) = 0$ and $\sigma_i(0) = \sigma_{i-1}(n_{i-1})$ for $2 \leq i \leq s$. From these difference equations, they derive an expression of the optimal value which can be computed in $O(s^2)$ time, thus proving that the evaluation version of the problem is solvable in polynomial time. However, even when $s = 1$, each job may have a different processing time on machine 1. So, here again, Clifford and Posner (2001) conclude that the optimization version of $Q2|pmtn|\sum_j C_j$ is in $EXP \setminus P$.

Note that, in view of the above description, every job j is preempted at most once in the optimal schedule, so that the job-oriented description $S_J(j)$ contains at most two elements for $j = 1, 2, \ldots, n$. We claim that $S_J(j)$ can be computed in polynomial time for all j, and hence, an optimal schedule can be generated with polynomial delay.

To establish our claim, consider a job j^*. Assume that j^* is the r-th replication of job type i^*. Job j^* starts on machine 1 as soon as all previous jobs have been completed on this machine, meaning at time $t_1 = \sum_{1 \le i < i^*} \sum_{1 \le j \le n_i} \sigma_i(j) + \sum_{1 \le j < r} \sigma_{i^*}(j)$. Standard summation formulas for power series allow to compute t_1 in polynomial time. We can also easily compute how much time j^* spends on machine 2 (namely, $(p_{i^*} - v_1\sigma_{i^*}(r))/v_2$ units of time) and, subtracting this quantity from t_1, deduce the starting time of j^* on machine 2. This yields a complete description of $S_J(j^*)$ in polynomial time. □

Example 3 **(**$F2|no-wait|CycleTime$**)**

This example deals with the problem of minimizing the cycle time of a set of jobs that is repeatedly produced in a no-wait two machine flowshop: in three-field scheduling notation, $F2|\textit{no-wait}|\textit{CycleTime}$. This problem is identical to the problem discussed in Section 1 with the additional requirement that, for each individual job, the operation on the second machine must start immediately after the completion of the operation on the first machine. It is one of the problems considered by Agnetis (1997), from which we borrow most of the analysis. The input takes the form $D = (s, n_1, n_2, \ldots, n_s, p_{11}, p_{12}, \ldots, p_{s1}, p_{s2})$. The single multiplicity version (in which $n_i = 1, i = 1, \ldots, s$) is solved by the well known algorithm for $F2|\textit{no-wait}|C_{\max}$ in Gilmore and Gomory (1964). We first briefly recall the polynomial algorithm for the single multiplicity version and subsequently deal with the high multiplicity problem.

Without loss of generality, assume that the job types are in increasing order of $p_{i2}, i = 1, \ldots, s$, the processing times on the second machine. Let π be a permutation of the job types in which the job types are in increasing order of $p_{i1}, i = 1, \ldots, s$, their processing times on the first machine. In the no-wait flowshop, a job j can only start on the first machine if its predecessor j' has left the first machine and has moved (without waiting) to the second machine. Then, job j can start processing on the first machine immediately if $p_{j1} \ge p_{j'2}$; otherwise it must wait for $p_{j'2} - p_{j1}$ time units, so that machine 2 is not occupied when job j terminates on machine 1. This no-wait problem is slightly different from, but equivalent to, the problem considered in Gilmore and Gomory (1964).

Now recall the cyclical context, where the last job in any cycle is followed by the first job of the next cycle. Obviously, the cycle time of a solution must

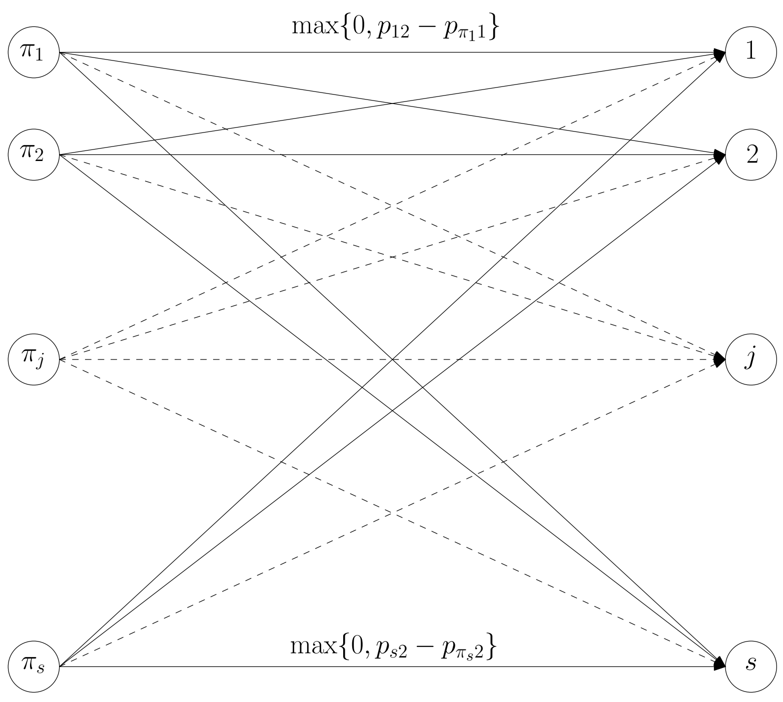
$\max\{0, p_{12} - p_{\pi_1 1}\}$
π_1
π_2
π_j
π_s
1
2
j
s
$\max\{0, p_{s2} - p_{\pi_s 2}\}$

Figure 1: Bipartite graph B

equal the sum of the processing times on the first machine and of the total idle time of the first machine, as it results from jobs waiting before they start in order to respect the no-wait requirement. The bipartite graph B in Figure 1 models the problem using this property. The nodes on the left hand side correspond to the jobs in order of the processing times on the first machine. The right hand side nodes correspond to the jobs in order of their index, i.e., in increasing order of processing times on the second machine. The cost of the arc, $c(\pi_j, j')$ between the left node π_j and the right node j' is set equal to the waiting time of the job corresponding to the left node, should it immediately succeed the job corresponding to the right node.

We now show that the value of a minimum-cost linear assignment in B forms a lower bound for the minimal cycle time. First notice that, given a feasible schedule, assigning jobs to their successors yields a feasible assignment. However not every feasible assignment implies a feasible set of successor relationships, and thus the value of the optimal linear assignment is indeed a lower bound. The cost of an assignment can be strictly lower than the value of the minimum cycle time, only if the arcs in the assignment induce more than one cycle in B. A simple example is the problem with two job types, each of single multiplicity, with $p_{11} = p_{12} = 1$, and $p_{21} = p_{22} = 2$. Here the optimal assignment has value zero, but the assignment corresponding to the single feasible schedule has value 1.

Gilmore and Gomory (1964) show that the linear assignment $\{(\pi_1, 1), (\pi_2, 2), \ldots, (\pi_s, s)\}$ forms a minimum cost linear assignment; see also Park (1991). Hence, the optimal linear assignment can be found by computing π, which is a sorting problem requiring $O(s \log s)$ time.

If the successor relationships defined by the optimal linear assignment do not form a single cycle in B, but partition the node set of B in components, the assignment can be modified to form a solution for the scheduling problem as follows Gilmore and Gomory (1964). Repeatedly find k such that nodes k and $k+1$ are in different components and $c(\pi_k, k+1) + c(\pi_{k+1}, k) - c(\pi_k, k) - c(\pi_{k+1}, k+1)$ is minimum. Interchange nodes π_k and π_{k+1}. Since B contains at most s components this step takes at most s times. Hence the complexity of the algorithm is $O(s \log s)$.

Now let us turn to the high multiplicity version of the problem. In this problem, the supply and demand of each of the nodes in B is set equal to the number of jobs of the corresponding type $i, i = 1, \ldots, s$. A general result of Hoffman (1963) implies that the classical Hitchcock transportation problem can be solved in the same greedy fashion, by an algorithm which has become known as the North-West Corner rule. The idea is simply to put an amount of flow on arc $(1, \pi_1)$ equal to the minimum of the supply of node π_1 and the demand of node 1. Subsequently delete nodes whose supply or demand is exhausted, and repeat. This procedure takes $O(s)$ time. The components of B can be merged together using the same simple greedy approach as in the single multiplicity version; after all, the transportation problem is the straightforward high multiplicity version of the linear assignment problem. Since again, there cannot be more than s components, this takes $O(s)$ time. Hence until here, the complexity is dominated by the $O(s \log s)$ sorting step.

This algorithm solves the decision version $\mathcal{SP}_1$ and the evaluation version $\mathcal{SP}_2$ of the high multiplicity $F2|\textit{no-wait}|\textit{CycleTime}$ problem in $O(s \log s)$ time. However, the required certificate is encoded implicitly, i.e., by specifying arc flows in the graph B rather than by giving starting and ending times of jobs. In other words, the optimization version $\mathcal{SP}_3$ is as yet not satisfactorily addressed. We now turn to this version.

A natural order to list the jobs in a solution for the high multiplicity version of $F2|\textit{no-wait}|\textit{CycleTime}$ is the processing order of the jobs, which is identical on both machines. Agnetis (1997) shows how the solution to the transportation problem, a set of flow values, can be turned into an implicit list of jobs. An optimal solution to the transportation problem can be assumed to have a positive flow on at most $2s$ arcs. Moreover, the greedy approach to turn this solution into a connected graph creates a positive flow on at most s additional arcs. Hence there are at most $3s$ arcs with positive flow. Together these arcs form a Eulerian multigraph, with contains a Eulerian cycle. Since the nodes correspond to jobs, any Eulerian cycle can be easily transformed into a sequence, a list, of jobs. Writing out this sequence explicitly requires exponential time and space, but Agnetis shows how to find a Eulerian cycle which is composed of at most s cycles, each of which naturally contains at most s nodes. A more formal and general treatment of this topic can be found in Grigoriev and Van De Klundert (2006). We conclude that it is possible to encode the optimal sequence compactly, i.e. polynomially, in space and time $O(s^2)$. It is left to the reader to verify that this compactly encoded sequence can be used to construct a list-generating algorithm that runs with polynomial delay. $\square$

Example 4 **($1|r_{ik}|C_{max}$)**

In this example, we explore the (machine, sequence)-oriented representation of a schedule for the high multiplicity version of $1|r_j|C_{max}$, which we denote by $1|r_{ik}|C_{max}$; see also Grigoriev (2003). In this high multiplicity version, we let $N_i = \sum_{j=1}^{i} n_j$, and $N_0 = 0$; so, jobs of type i are indexed as $N_{i-1}+1, \ldots, N_i$. We assume that the release date r_{ik} of job k (of type i) is defined as $r_{ik} = a_i + kT_i$, where a_i, T_i are input parameters, for $i = 1, \ldots, s$, $k = N_{i-1}+1, \ldots, N_i$.

It is well known that processing the jobs by nondecreasing release dates (*Earliest Release Date*, or ERD rule) yields an optimal sequence. Hence the optimal sequence, and therefore a (machine, sequence)- oriented description of an optimal schedule for $1|r_j|C_{max}$ can be found in $O(s \log s)$ time. Moreover $\mathcal{SP}_1$, $\mathcal{SP}_2$ and $\mathcal{SP}_3$ can be solved in polynomial time for the single multiplicity version. Of course, the same ERD sequence is also optimal for the high multiplicity version and a (machine, sequence)-oriented representation of the ERD schedule can be constructed in polynomial time as follows. (In the remainder we require that jobs sharing a same release date are sequenced in increasing order of their index.)

Consider an individual job k of type i. For each job type $j, j = 1, \ldots, s$, let k_j be the smallest integer such that $N_{j-1}+1 \leq k_j \leq N_j$ and $a_i + kT_i \leq a_j + k_jT_j$. Redefine $k_j := k_j + 1$ if $a_i + kT_i = a_j + k_jT_j$ and $i \geq j$. Then, by definition, k_j is the first individual job of type j following job k (of type i) in the ERD schedule. Let k_{j^*} be the job with minimum index whose release date equals $\min_{j=1,\ldots,s} a_j + k_jT_j$. Setting $S(1,k) = k_{j^*}$ if k_j^* exists and 0 otherwise yields a (machine, sequence)-oriented representation of the ERD schedule. Clearly, S can be evaluated in polynomial time. Thus, a (machine, sequence)-oriented representation of the ERD schedule for $\mathcal{SP}_3$ can be computed in polynomial time. Likewise, a polynomial delay list generating algorithm is easily constructed. On the other hand, we do not know the complexity of $\mathcal{SP}_1$ and $\mathcal{SP}_2$ for this problem. $\square$

	$\mathcal{SP}_1$	$\mathcal{SP}_2$	$\mathcal{SP}_3$	
			List-generating	Pointwise
$1\|p_j = 1\|\sum_j w_j U_j$	P	P	polynomial delay	S_J, S_T polynomial
total deviation JIT	?	?	total polynomial	?
max deviation JIT	co-NP	?	total polynomial	?
max deviation JIT, fixed s	P	P	polynomial delay	?
$P\|pmtn\|C_{max}$	P	P	polynomial delay	S_J polynomial delay, S_{MT} polynomial
$Q2\|pmtn\|\sum_j C_j$	P	P	polynomial delay	S_J polynomial
$F2\|no\text{-}wait\|CycleTime$	P	P	polynomial delay	S_{MS} polynomial
$1\|r_{ik_i}\|C_{max}$	?	?	polynomial delay	S_{MS} polynomial

Table 1: Complexity of various problems

The results concerning the different models discussed in this section, and in Brauner et al. (2005), are summarized in Table 1. Note that all these problems can be solved in pseudo-polynomial time. A question mark in the table means that we do not know anything beyond this fact (which is often nontrivial in itself).

6 Summary and conclusions

High multiplicity scheduling problems are commonly encountered in innovative real life settings, be it in automated manufacturing or in telecommunication applications. They have been widely investigated and classified using traditional methods and complexity classes. Many of the results and problems have not been appropriately addressed in this context, and hence we proposed in Brauner et al. (2005) a first classification scheme for high multiplicity problems, restricted to single machine non-preemptive problems. This paper considers multiple machine preemptive settings, and the analysis and the applications demonstrate that the framework fits this more general context equally well.

The framework provides a refined notion of what it means to "solve a high multiplicity scheduling problem", and allows to classify various problems and results. Nevertheless a number of problems remain open, for instance the 3-machine flowshop problem with 2 job types mentioned in the prelude. Grigoriev (2003) provides more results on high multiplicity scheduling and a list of several open problems.

The results displayed in Table 1 suggest that the relationship between different complexity classes may go deeper than the simple implications mentioned in

Propositions 1 or 2. It would be useful to investigate some of these relations in future work, for instance by identifying problems which are complete for their respective classes.

References

Agnetis A. (1997), No-wait flowshop scheduling with large lot size, *Annals of Operations Research* **70**, 415–438.

Bar-Noy A., R. Bhatia, J.S. Naor and B. Schiber (2002), Minimizing service and operation costs of periodic scheduling, *Mathematics of Operations Research* **27**, 518–544.

Brauner N. and Y. Crama (2004), The maximum deviation just-in-time scheduling problem, *Discrete Applied Mathematics* **134**, 25–50.

Brauner N., Y. Crama, A. Grigoriev, J. Van De Klundert (2005), A framework for the complexity of high-multiplicity scheduling problems, *Journal of Combinatorial Optimization* **9**, 313–323.

Brauner N., G. Finke and W. Kubiak (2003), Complexity of one-cycle robotic flow-shops, *Journal of Scheduling* **6**, 355–371.

Clifford J.J. and M. E. Posner (2000), High multiplicity in earliness-tardiness scheduling, *Operations Research* **48**, 788–800.

Clifford J.J. and M.E. Posner (2001), Parallel machine scheduling with high multiplicity, *Mathematical Programming* **89**, 359–383.

Cosmadakis S.S. and C.H. Papadimitriou (1984), The traveling salesman problem with many visits to few cities, *SIAM Journal on Computing* **13**, 99–108.

Dyer M.E. (1983), The complexity of vertex enumeration methods, *Mathematics of Operations Research* **8**, 381–402.

Garey, M.R. and D.S. Johnson (1979), *Computers and intractability: A guide to the theory of NP-completeness*, Freeman, San Francisco.

Garey M.R., D.S. Johnson and R. Sethi (1976), The complexity of flowshop and jobshop scheduling, *Mathematics of Operations Research* **1**, 117–129.

Gilmore P.C. and R.E. Gomory (1964), Sequencing a one-state variable machine: A solvable case of the traveling salesman problem, *Journal of Operations Research Society of America* **12**, 655–679.

Granot F. and J. Skorin-Kapov (1993), On polynomial solvability of the high multiplicity total weighted tardiness problem, *Discrete Applied Mathematics* **41**, 139–146.

Grigoriev A. (2003), *High multiplicity scheduling problems*, Doctoral thesis, Maastricht University, Maastricht, The Netherlands.

Grigoriev A. and J. Van De Klundert (2006), On the high multiplicity traveling salesman problem. *Discrete Optimization* **3**, 50–62.

Hochbaum D.S. and R. Shamir (1990), Minimizing the number of tardy job units under release time constraints, *Discrete Applied Mathematics* **28**, 45–57.

Hochbaum D.S. and R. Shamir (1991), Strongly polynomial algorithms for the high multiplicity scheduling problem, *Operations Research* **39**, 648–653.

Hochbaum D.S., R. Shamir and J.G. Shanthikumar (1992), A polynomial algorithm for an integer quadratic nonseparable transportation problem, *Mathematical Programming* **55**, 359–376.

Hoffman A.J. (1963), On simple linear programming problems, in *Proc. Symposium on Pure Mathematics* **7**, AMS, Providence, 317-327.

Hurink J. and S. Knust (2001), Makespan minimization for flow-shop problems with transportation times and a single robot, *Discrete Applied Mathematics* **112**, 199–216.

Johnson S.M. (1954), Optimal two- and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly* **1**, 61–68.

Johnson D.S., M. Yannakakis and C.H. Papadimitriou (1988), On generating all maximal independent sets, *Information Processing Letters* **27**, 119–123.

Kubiak W. and S.P. Sethi (1991), A note on "Level schedules for mixed-model assembly lines in just-in-time production systems", *Management Science* **37**, 121–122.

Kubiak W. and S.P. Sethi (1994), Optimal just-in-time schedules for flexible transfer lines, *International Journal of Flexible Manufacturing Systems* **6**, 137–154.

Lawler E.L., J.K. Lenstra and A.H.G. Rinnooy Kan (1980), Generating all maximal independent sets: NP-hardness and polynomial-time algorithms, *SIAM Journal on Computing* **9**, 558–565.

McCormick S.T., S.R. Smallwood and F.C.R. Spieksma (2001), A polynomial algorithm for multiprocessor scheduling with two job lengths, *Mathematics of Operations Research* **26**, 31–49.

McNaughton R. (1959), Scheduling with deadlines and loss functions. *Management Science* **6**, 1–12.

Miltenburg J. (1989), Level schedules for mixed-model assembly lines in just-in-time production systems, *Management Science* **35**, 192–207.

Papadimitriou, C.H. and K. Steiglitz (1982), *Combinatorial optimization: Algorithms and complexity*, Prentice Hall: Englewood Cliffs, New Jersey.

Park J.K. (1991), A special case of the n-vertex traveling salesman problem that can be solved in $O(n)$ time, *Information Processing Letters* **40**, 247–254.

Pinedo M. (1995), *Scheduling: Theory, algorithms and systems*, Prentice Hall: Englewood Cliffs, New Jersey.

Posner M.E. (1985), *The complexity of earliness and tardiness scheduling problems under id-encoding*, Working Paper 85-70, New York University, New York.

Psaraftis H.N. (1980), A dynamic programming approach for sequencing groups of identical jobs, *Operations Research* **28**, 1347–1359.

Rothkopf M. (1966), The travelling salesman problem: On the reduction of certain large problems to smaller ones, *Operations Research* **14**, 532–533.

Steiner G. and J.S. Yeomans (1993), Level schedules for mixed-model, just-in-time processes, *Management Science* **39**, 728–735.

T'kindt V., K. Bouibede-Hocine and C. Esswein (2005), Counting and enumeration complexity with application to multicriteria scheduling, *4OR* **3**, 1–21.