# On the complexity of high multiplicity scheduling problems

Nadia Brauner, Y. Crama, A. Grigoriev, J. van de Klundert

# On the complexity of high multiplicity scheduling problems

**N. Brauner[1], Y. Crama[2], A. Grigoriev[3], J. van de Klundert[4]**

October 2001

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

[2]Ecole d'Administration des Affaires, University of Liège, Boulevard du Rectorat 7 (B31), 4000 Liège, Belgium, Y.Crama@ulg.ac.be

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

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

**Abstract**

The purpose of this note is to propose a definition of several complexity classes which could prove useful for the analysis of high multiplicity scheduling problems. Part of this framework relies on previous work, aiming at the definition of output-sensitive complexity measures for the analysis of algorithms producing "large" outputs. However, the classes differ according as we look at schedules as sets of processing intervals, or as related (single-valued or set-valued) mappings.

**Acknowledgements.** The authors are grateful to Tom McCormick, Maurice Queyranne and Gerhard Woeginger for their pointed comments on a preliminary version of this note. This work was carried out while the first author was visiting the University of Liège in the framework of a postdoctoral project supported by the European Network DONET (contract number ERB TMRX-CT98-0202). The second author acknowledges the partial financial support of ONR (grant N00014-92-J-1375), NSF (grant DMS-98-06389), and research grants from the Natural Sciences and Engineering Research Council of Canada (NSERC). The fourth author acknowledges financial support from the APPOL project (EU - IST 1999 14084).

# 1 Introduction

The purpose of this note is to propose a definition of several complexity classes which could prove useful for the analysis of so-called *high multiplicity scheduling problems*. Such problems have been recently investigated by several researchers (see e.g. Rothkopf (1966), Psaraftis (1980), Cosmadakis and Papadimitriou (1984), Posner (1985) for early references, and other articles cited below for more recent ones). Hochbaum and Shamir (1990, 1991), in particular, have coined the term "high multiplicity" and have underlined the need to discuss the complexity of such problems with special care. A paper by Clifford and Posner (2001) provides a more detailed framework for this complexity analysis, as well as applications to several specific problems.

We take a further step along this same line of research, by formulating several proposals to cast high multiplicity scheduling problems into a more precise computational complexity framework. Sections 2 and 3 describe the class of scheduling problems that we want to consider. Section 4 contains a typology of algorithms which is applicable in the simplest case of non-preemptive one-machine scheduling problems. These concepts are illustrated on specific examples in Section 5. The typology is extended to more general scheduling problems (involving multiple machines and job preemptions) in Section 6, and further illustrated in Section 7. The paper concludes with a brief discussion and an outline of perspectives for further research.

# 2 High multiplicity scheduling problems

For the sake of simplicity, we initially restrict ourselves to non-preemptive one-machine scheduling problems. More complex problem formulations will be tackled in Section 6.

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 $p_j$, its release date $r_j$, its due date $d_j$, etc. The binary input size of an instance of $\mathcal{SP}$ is $O(nL)$, where $L$ is the largest input size of an attribute.

It frequently happens, however, that the input of a scheduling problem can be described in a much more compact way, due to the fact that the jobs naturally fall into a small number, say $s << n$, of distinct *job types*, where all the jobs of a same type share exactly the same characteristics, i.e. attribute values. When this is the case, we only need to describe *one* representative job in order to completely define a type, so that an instance of the problem $\mathcal{SP}$ only 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 $i$.

When the data is encoded in this compact form, we say that $\mathcal{SP}$ is a *high multiplicity* scheduling problem. So, a generic instance of the (one-machine non-preemptive) high multiplicity scheduling problem $\mathcal{SP}$ takes the form $D = (s, n_1, n_2, \ldots, n_s, \Delta)$, where $\Delta$ comprises all the relevant job attributes.

This kind of situation is encountered for instance in repetitive manufacturing environments. Here, a *minimal part set* (MPS) is described by a vector $(n_1, n_2, \ldots, n_s)$ where $n_i$ represents the number of parts of type $i$ in the MPS ($1 \leq i \leq s$). The whole part set is then viewed as consisting of a large number of copies of the MPS (in the limit, infinitely many copies) to be produced repeatedly (see e.g. Miltenburg (1989) and Pinedo (1995)).

In other applications, the number of job types may be artificially reduced by aggregating jobs with different, but similar characteristics, into a single type. The resulting scheduling problem is only an approximation of the original one, but may prove easier to handle (Hochbaum et al. 1992).

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. 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. This distinction has been drawn by several authors and stressed especially by Hochbaum and Shamir (1990, 1991), Hochbaum, Shamir, and Shanthikumar (1992), Clifford and Posner (2001).

In order to discuss it more precisely, we need a formal definition of the problems we are dealing with.

# 3 A scheduling problem is three problems

Consider an instance $D = (s, n_1, n_2, \ldots, n_s, \Delta)$ of a (one-machine non-preemptive) high multiplicity scheduling problem $\mathcal{SP}$. 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.

A schedule for the instance $D$ can be seen as an assignment $S^* : \{1, 2, \ldots, n\} \to \mathbb{R}$ where $S^*(j)$ denotes the starting time of job $j$ ($j = 1, 2, \ldots, n$). However, in order to be able to handle more general problem formulations (in Section 6), and at the risk of appearing fussy, we prefer to define a *schedule* for an instance $D$ as a (finite) *subset* $S$ of $\{1, 2, \ldots, n\} \times \mathbb{R} \times \mathbb{R}$, where $S$ may be (and usually, is) restricted to belong to a set $\mathcal{F}_D$ of *feasible* schedules associated with $D$. The interpretation of $S$ is that, if $(j, t_1, t_2) \in S$, then job $j$ is processed continuously during the time interval $[t_1, t_2]$. This model allows to recover more traditional readings of the schedule by considering various *mappings* derived from $S$, for instance:

- a *job-oriented* description of the schedule is defined by the mapping

$$S_J : \{1, 2, \ldots, n\} \to \mathbb{R} \times \mathbb{R} : j \mapsto S_J(j) = (t_1, t_2) \text{ if } (j, t_1, t_2) \in S;$$

- a *time-oriented* description of the schedule is defined by the mapping

$$\begin{aligned} S_T : \mathbb{R} \to \{0, 1, \ldots, n\} \times \mathbb{R} \times \mathbb{R} : t \mapsto S_T(t) &= (j, t_1, t_2) && \text{if } (j, t_1, t_2) \in S \text{ and job } j \text{ is} \\ & && \text{the first job to be processed} \\ & && \text{at or after time } t, \\ &= (0,0,0) && \text{if there are no more jobs to} \\ & && \text{be processed after time } t. \end{aligned}$$

In the one-machine non-preemptive context, the job-oriented description $S_J$ is roughly equivalent to the full description of $S$, as well as to the mapping $S^*$ mentioned earlier. The 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.

Of course, still numerous other mappings could be associated with the schedule $S$. For instance, a mapping from time instants to *job types*, rather than to individual jobs, may prove useful in some frameworks. But for the sake of our discussion, we shall limit ourselves to the above-mentioned mappings.

Let us now turn to the objective function of $\mathcal{SP}$. If $\mathcal{F}_D$ denotes again the set of feasible schedules associated with $D$, we let $f_D \colon \mathcal{F}_D \to \mathbb{R}$ be the cost 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 $D$, and that $f_D$ always attains its minimum over $\mathcal{F}_D$. Moreover, we also 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 $|S|$ (note that this is a rather weak hypothesis).

As in Papadimitriou and Steiglitz (1982) (where we borrowed the title of this section), we now define three distinct scheduling problems associated with $\mathcal{F}_D$ and $f_D$ (see also Clifford and Posner (2001)).

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, analyzing the complexity of any specific high multiplicity problem may turn out to be a tricky matter. Indeed, proving that $\mathcal{SP}_1$ is in $NP$, for instance, requires the existence of an algorithm $\mathcal{A}$ and of a polynomial-size *certificate* $c(D, K)$ for each Yes-instance $(D, K)$ of $\mathcal{SP}_1$, with the property that, when applied to $c(D, K)$, $\mathcal{A}$ returns the answer Yes after a polynomial number of steps (we use the terminology of Papadimitriou and Steiglitz (1982)). Intuitively, when the answer to $\mathcal{SP}_1$ is affirmative, the certificate provides a concise proof that it is indeed so. Now, the most natural certificate for problem $\mathcal{SP}_1$ would be a feasible schedule $S$ such that $f_D(S) \leq K$. But in many cases, obvious descriptions of $S$ are not concise, i.e. not polynomial in the size of $(D, K)$. (As a matter of fact, it may even be the case that some high multiplicity recognition problems are neither in $NP$ nor in co-$NP$.) Similar problems pop up, of course, if we are to prove that $\mathcal{SP}_1$ or $\mathcal{SP}_2$ is in $P$.

In spite of these difficulties, many high multiplicity scheduling problems have been

proved to be polynomially solvable (see for instance Brauner et al. (2000), Clifford and Posner (2001, 2000), Granot and Skorin-Kapov (1993), Hochbaum and Shamir (1990, 1991), Hochbaum et al. (1992), Hurink and Knust (1998), McCormick et al. (1993), etc.) or in co-$NP$ (Brauner and Crama (2000)) or $NP$-hard (Clifford and Posner (2000, 2001), Posner (1985), Bar-Noy et al. (1998), etc.). 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 $O(s \log n + sL)$. The certificates, clearly, do not enumerate the list of $n$ starting times, but rather provide an implicit, concise encoding of these starting times. Let us illustrate this on a problem solved by Hochbaum and Shamir (1991).

**Example 1 (Weighted number of tardy jobs).** In the classical three-field notation, this is the problem $1/p_j = 1/\sum_j w_j U_j$. Its input takes the form

$$D = (s, n_1, n_2, \ldots, n_s, d_1, d_2, \ldots, d_s, w_1, w_2, \ldots, w_s),$$

where $d_i$ is the due-date for jobs of type $i$ and $w_i$ is their weight. All jobs are assumed to have unit-processing time. The objective function is to minimize the weighted number of tardy jobs. Hochbaum and Shamir proved that the problem can be transformed into a transportation problem of dimension $s \times (s+1)$, where variable $x_{it}$ indicates the number of jobs of type $i$ processed in the interval $(d_{t-1}, d_t]$, for $i = 1, 2, \ldots, s$, $t = 1, 2, \ldots, s+1$ (wolog, $d_0 = 0 \leq d_1 \leq \ldots \leq d_s \leq d_{s+1} = n$). The variables must satisfy the transportation constraints

$$\begin{array}{rll} \sum_{i=1}^{s} x_{it} & = & d_t - d_{t-1}, \qquad t = 1, 2, \ldots, s+1; \\ \sum_{t=1}^{s+1} x_{it} & = & n_i, \qquad\quad i = 1, 2, \ldots, s. \end{array}$$

Consequently, the recognition and the evaluation version of this problem can be solved in (strongly) polynomial time (in fact, in $O(s \log s)$ time if a specialized greedy algorithm is used). □

Let us now turn to the optimization problem $\mathcal{SP}_3$. Few authors have attempted to discuss precisely what it means to "solve" $\mathcal{SP}_3$. Namely, the way in which the optimal schedule $S$ should be described, is usually not explicitly stated.

Hochbaum and Shamir (1991) and Hochbaum et al. (1992) have observed that $\mathcal{SP}_3$ can sometimes be solved by first obtaining a concise encoding of the optimal schedule, then applying a decoding algorithm to generate all the elements of the schedule. For instance, in Example 1 above, the solution $(x_{it})$ of the transportation problem provides a concise encoding of the solution. In order to obtain a schedule in extension, i.e. in order to compute a starting time for each job, one needs to "decode" the solution $(x_{it})$ by carrying out additional computations (see Section 5).

Clifford and Posner (2001) established a clear distinction between the recognition version and the optimization version of high multiplicity problems. They noticed: "Under high multiplicity encoding, we cannot output a job-by-job, machine-by-machine schedule if we want the output length to be a polynomial function of the input length. Consequently, we allow for a high multiplicity description of the schedule." Clifford and Posner set out to define a *job group* as an arbitrary subset of jobs of a particular type and describe a high multiplicity schedule as an ordered set of job groups (their definition is actually complicated by the fact that they consider multiple machines). In Clifford and Posner (2001), only this type of schedule is regarded as "legal". Under this hypothesis, the authors argue that certain high multiplicity problems are in $EXP \setminus P$, meaning that such problems are solvable in exponential time, but not in polynomial time. More precisely, they establish that there is no polynomial description of the optimal schedule in terms of job groups.

This view of high multiplicity schedules, however, may appear to be overly restrictive. In the next section, we propose more general models for describing a solution of problem $\mathcal{SP}_3$. An advantage of these alternate models is that they allow to obtain a more precise complexity classification of algorithms solving $\mathcal{SP}_3$.

# 4 Complexity models

In this section, we shall rely on several interpretations of the task "output an optimal schedule $S$". A main distinction takes place according as we view schedules as sets, or as we focus on one of their derived (single-valued) mappings.

## 4.1 List-generating algorithms

In our first interpretation, we assume that the set $S$ is to be generated in extension.

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

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

The classification of list-generating algorithms to be described in Definition 2 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 Lawler, Lenstra, and Rinnooy Kan (1980) or Dyer (1983) for related concepts).

**Definition 2.** A list-generating algorithm $\mathcal{A}$ for $\mathcal{SP}_3$ runs in:

- *pseudo-polynomial time* if $\tau(n)$ is polynomially bounded in $|D|$ and $M$, where $M$ is the largest number appearing in $D$;
- *polynomial total time* if $\tau(n)$ is polynomially bounded in $n$ and $|D|$;
- *incremental polynomial time* if $\tau(j) - \tau(j-1)$ is polynomially bounded in $j$ and $|D|$, for $j = 1, 2, \ldots, n$;
- *polynomial delay* if $\tau(j) - \tau(j-1)$ is polynomially bounded in $|D|$, for $j = 1, 2, \ldots, n$;
- *polynomial time* if $\tau(n)$ is polynomially bounded in $|D|$.

These definitions will be illustrated on several scheduling problems in Section 5. For now, let us place a few comments on the above definitions.

Pseudo-polynomiality and polynomiality are the usual concepts from complexity theory and are only mentioned here for the sake of completeness. In particular, if $\mathcal{SP}_3$ can be solved in polynomial time, then $n$ is 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 $|S|$).

Total time polynomiality 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 the starting times of all $n$ jobs must grow at least linearly with $n$.

Incremental polynomial time captures the idea that the algorithm outputs the starting times sequentially and does not spend "too much time" between two successive outputs. In computing the starting time of job $\pi^j$, however, the algorithm may need to look at the starting times of $\pi^1, \pi^2, \ldots, \pi^{j-1}$ (for instance, to check feasibility of the partial schedule) and therefore we allow $\tau(j) - \tau(j-1)$ to depend on $j$ as well as on $|D|$.

**Remark 1.** The intuition behind this concept is slightly different from that in Johnson et al. (1988). A more straightforward adaptation of the definition proposed by Johnson et al. would go something like this: given *any* subset of jobs, say $J \subset \{1, 2, \ldots, n\}$, as well as the starting times of all the jobs in $J$, the algorithm should be able to output a job $k \notin J$ and its processing interval $[t_1^k, t_2^k]$ in time polynomially bounded in $|J|$ and $|D|$. This notion is stronger than the one we introduced, but we do not feel that it presents much interest in the scheduling context. □

**Remark 2.** Johnson et al. (1988) also consider the case where $\mathcal{A}$ is required to produce its outputs in some specified order. Such a requirement may easily be adapted for scheduling problems. For instance, since schedules are to be implemented in the course of time, it may appear "natural" to ask for $\mathcal{A}$ to output $S_J(1), S_J(2), \ldots, S_J(n)$ in increasing order of the starting times. Whether such requirements actually are meaningful or not may be debatable. At any rate, the corresponding algorithms can still be analyzed in terms of the above classification. □

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.

The following statement summarizes the above discussion.

**Proposition 1** *If $\mathcal{A}$ is a list-generating algorithm for the optimization version $\mathcal{SP}_3$ of a single-machine scheduling problem without preemptions, then:*

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

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

Moreover, if $\mathcal{A}$ runs in polynomial total time, then $\mathcal{A}$ also runs in pseudo-polynomial time, since $n = \sum_{1 \leq j \leq s} n_j \leq M^2$ by definition of $M$. □

## 4.2 Pointwise job-oriented algorithms

In this and the next section, we assume that the algorithm $\mathcal{A}$ is not necessarily required to produce the optimal schedule in extension, but that it should only be able to compute one of the mappings derived from $S$ as explained in Section 3.

**Definition 3.** A *pointwise job-oriented algorithm* for $\mathcal{SP}_3$ is an algorithm $\mathcal{A}$ which, on every input $(D, j)$ ($j \in \{1, 2, \ldots, n\}$), outputs a pair $S_J(j) = (t_1(j), t_2(j))$ such that

$$S = \{ (j, t_1(j), t_2(j)) : j \in \{1, 2, \ldots, n\} \}$$

is an optimal schedule for $D$.

As usual, a pointwise job-oriented algorithm for $\mathcal{SP}_3$ is polynomial if, for every instance $D$ of $\mathcal{SP}_3$ and every $j \in \{1, 2, \ldots, n\}$, $\mathcal{A}$ outputs $S_J(j)$ in time polynomially bounded in $|D|$. Thus, the existence of a polynomial pointwise algorithm $\mathcal{A}$ simply means that the job-oriented description of the optimal schedule can be queried in polynomial time or, in other words, that the function $S_J : \{1, 2, \ldots, n\} \rightarrow \mathbb{R} \times \mathbb{R}$ can be computed in polynomial time (in the sense of Garey and Johnson (1979)).

The following relations hold.

**Proposition 2** *For a single-machine scheduling problem without preemptions,*
(a) *if $\mathcal{SP}_3$ has a polynomial list-generating algorithm, then $\mathcal{SP}_3$ has a polynomial pointwise job-oriented algorithm;*
(b) *if $\mathcal{SP}_3$ has a polynomial pointwise job-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 job-oriented algorithm, then $\mathcal{A}$ can be called $n$ times to compute successively $S_J(1), S_J(2), \ldots, S_J(n)$. Since the running time of each call is polynomial in $|D|$, the resulting list-generating algorithm runs with polynomial delay. □

So, intuitively, polynomial pointwise algorithms fall somewhere between polynomial and polynomial-delay list-generating algorithms in the hierarchy described in Proposition 1.

We shall present some example of polynomial pointwise algorithms in Section 5. Interestingly, all such algorithms actually consist of two distinct algorithms: a first algorithm

$\mathcal{A}_e$ which solves $\mathcal{SP}_2$ while producing a compact "encoding" ("certificate") $\Sigma$ of the optimal schedule $S$, and a second algorithm $\mathcal{A}_u$ which computes $S_J$ by "decoding" the output produced by $\mathcal{A}_e$. Let us formulate this notion in more precise terms.

**Definition 4.** A 2-*phase job-oriented algorithm* for $\mathcal{SP}_3$ is a pair of algorithms $(\mathcal{A}_e, \mathcal{A}_u)$ such that
1) on the input $D$, $\mathcal{A}_e$ outputs a string $(f_D^{opt}, \Sigma)$ where $f_D^{opt}$ is the optimal objective value of $\mathcal{SP}_3$;
2) on the input $(D, j, f_D^{opt}, \Sigma)$, $\mathcal{A}_u$ outputs a pair $S_J(j) = (t_1(j), t_2(j))$ such that

$$S = \{ (j, t_1(j), t_2(j)) : j \in \{1, 2, \ldots, n\} \}$$

is an optimal schedule for $D$.

The string $\Sigma$ in this definition represents the encoding of the optimal solution.

Now, a 2-phase job-oriented algorithm runs in polynomial time if both $\mathcal{A}_e$ and $\mathcal{A}_u$ run in time polynomial in the size of their respective inputs (this implies, in particular, that the size of $\Sigma$ must be polynomially related to the size of $D$). Note that these requirements are stronger than the requirements for a polynomial pointwise job-oriented algorithm: indeed, for a polynomial pointwise job-oriented algorithm to be polynomial, it need not compute the optimal objective value $f_D^{opt}$ in time polynomial in $|D|$. In fact, for such an algorithm, the only requirement on the computation of the objective function value comes from our blanket assumption that, given a description of $S$ in extension, the corresponding objective function value can be computed in time polynomial in $|D|$ and $|S|$ (cf. Section 3). From a complexity viewpoint, the following problem captures the difference between polynomial 2-phase job-oriented algorithms and polynomial pointwise job-oriented algorithms:

3SAT SCHEDULING:
INSTANCE: $D = (2^N, C)$, where $C$ is a 3SAT instance on $N$ variables, and there are $n = 2^N$ jobs of the same type.
SCHEDULES: There is only one feasible schedule, namely $S = \{(j, j, j+1) : j \in \{1, 2, \ldots, 2^N\}\}$.
OBJECTIVE FUNCTION: $f_D = 1$ if $C$ is satisfiable, $f_D = 0$ otherwise.

The reader may verify that all of our blanket assumptions concerning high multiplicity problems hold for 3SAT SCHEDULING; in particular, the optimal value of the schedule can be computed in time polynomial in the length of the schedule, viz. $2^N$.

**Proposition 3** *Problem* 3SAT SCHEDULING *has a polynomial pointwise job-oriented algorithm, but it has no polynomial 2-phase job-oriented algorithm unless* $P = NP$.

**Proof** For 3SAT SCHEDULING, the mapping $S_J$ can trivially be computed in polynomial time: given $(D, j)$, just return $(j, j+1)$. Hence the problem has a polynomial pointwise job-oriented algorithm. But computing the optimal value of the problem (i.e., solving $\mathcal{SP}_2$) is NP-hard, since this amounts to solving the 3SAT instance. Since a 2-phase job-oriented algorithm requires an algorithm $\mathcal{A}_e$ which outputs a string $(f_D^{opt}, \Sigma)$ where $f_D^{opt}$ is the optimal objective value of $\mathcal{SP}_3$, such an algorithm cannot be polynomial unless $P = NP$. □

The following proposition identifies the conditions under which the existence of a polynomial 2-phase job-oriented algorithm is equivalent to the existence of a pointwise job-oriented algorithm.

**Proposition 4** *The optimization version $\mathcal{SP}_3$ of a single-machine scheduling problem without preemptions has a polynomial 2-phase job-oriented algorithm if and only if it has a polynomial pointwise job-oriented algorithm and the corresponding evaluation problem $\mathcal{SP}_2$ is polynomially solvable.*

**Proof** If $\mathcal{SP}_3$ has a polynomial 2-phase job-oriented algorithm $(\mathcal{A}_e, \mathcal{A}_u)$, then a polynomial pointwise job-oriented algorithm is obtained as follows. When handed the input $(D, j)$ with $j \geq 1$, the algorithm first runs $\mathcal{A}_e$ on $D$ to obtain $(f_D^{opt}, \Sigma)$, then runs $\mathcal{A}_u$ on $(D, j, f_D^{opt}, \Sigma)$ to compute $S_J(j)$. Note that the total running time of this procedure is polynomial in $|D|$. Moreover, $\mathcal{A}_e$ solves $\mathcal{SP}_2$ in polynomial time.

Conversely, let $\mathcal{A}^*$ be an algorithm that solves $\mathcal{SP}_2$ in polynomial time. If $\mathcal{SP}_3$ has a polynomial pointwise job-oriented algorithm $\mathcal{A}$, then identify $\mathcal{A}_e$ with $\mathcal{A}^*$. The output of $\mathcal{A}^*$ is simply $f_D^{opt}$, thus the string $\Sigma$ is empty. The decoding algorithm $\mathcal{A}_u$ can be identified with $\mathcal{A}$: when running on the input $(D, j, f_D^{opt})$, the algorithm simply ignores the information $f_D^{opt}$. □

By definition, if $\mathcal{SP}_2$ is polynomially solvable, then $f_D^{opt}$ can be computed in time polynomial in $|D|$. This property is clearly different from our general assumption for high multiplicity scheduling problems, namely that given a description of $S$ in extension, $f_D(S)$ can be computed in time polynomial in $|D|$ and $|S|$. This difference is responsible for the two different results stated in Proposition 4 and Proposition 3 above.

**Remark 3.** The reader may note that there remains some room for an intermediate result between Proposition 4 and Proposition 3. Indeed, it is open whether polynomial 2-phase job-oriented algorithms are equivalent to polynomial pointwise job-oriented algorithms under the following condition: given a description of an optimal schedule

$S$, $f_D(S)$ can be computed in time polynomial in $D$. This condition implicitly requires that the description of an optimal schedule is not the extensive list of jobs, since such a list is by definition of superpolynomial length. However, it does not require that the optimal schedule $S$ be found in time polynomial in $D$, nor does it require that $\mathcal{SP}_2$ be polynomially solvable. □

## 4.3 Pointwise time-oriented algorithms

Of course, a definition similar to Definition 3 can be proposed for time-oriented descriptions of the optimal schedule.

**Definition 5.** A *pointwise time-oriented algorithm* for $\mathcal{SP}_3$ is an algorithm $\mathcal{A}$ which, on the input $(D, t)$ with $t \in \mathbb{R}$, outputs $S_T(t)$, where $S_T$ is the time-oriented mapping derived from an optimal schedule $S$.

Similarly to Proposition 2, the following relations hold.

**Proposition 5** *For a single-machine scheduling problem without preemptions,*
(a) *if $\mathcal{SP}_3$ has a polynomial list-generating algorithm, then $\mathcal{SP}_3$ has a polynomial pointwise time-oriented algorithm;*
(b) *if $\mathcal{SP}_3$ has a polynomial pointwise time-oriented algorithm, and if an upper-bound $U$ on the makespan of the optimal schedule can be computed in polynomial time, then $\mathcal{SP}_3$ has a polynomial-delay list-generating algorithm.*

**Proof** Assertion (a) holds as in Proposition 2.

Conversely, if $\mathcal{A}$ is a polynomial pointwise time-oriented algorithm, then $\mathcal{A}$ can be used to determine the makespan of $S$, by binary search over the interval $[0, U]$ (note that $\mathcal{A}$ returns $S_T(t) = (0, 0, 0)$ if and only if $t$ exceeds the makespan of the optimal schedule). This requires $O(\log U)$ calls on $\mathcal{A}$, where $\log U$ is polynomial in $|D|$. Say the makespan is equal to $C_{max}$, and $S_T(C_{max}) = (j, t_1, t_2)$. Then, $j$ is the last job to be scheduled. The same procedure can be iterated over the interval $[0, t_1]$, and eventually generates the processing intervals of all the jobs in reverse order. The whole procedure runs with polynomial delay. □

# 5 Applications: single-machine models

We now illustrate the concepts introduced in the previous sections on several examples of high multiplicity problems.

**Example 1 (Weighted number of tardy jobs – continued).** A discussion of this problem was already started in Section 3. Let us now show that the approach in Hochbaum and Shamir (1991) leads to a polynomial two-phase job-oriented algorithm (and hence, to a polynomial pointwise job-oriented algorithm) for the optimization version of this problem. First, the solution $(x_{it})$ of the transportation problem can be computed in $O(s \log s)$ time and constitutes the required encoding $\Sigma$. Then, given a job index $j$, we first determine the type of this job, i.e. the unique index $i^*$ such that $\sum_{1 \leq i < i^*} n_i < j \leq \sum_{1 \leq i \leq i^*} n_i$. If $r = j - \sum_{1 \leq i < i^*} n_i$, we look at job $j$ as the $r$-th replication of job type $i^*$. Next, we compute the index of the interval where $j$ must be scheduled: this is the value of $t^*$ such that $\sum_{1 \leq t < t^*} x_{i^* t} < r \leq \sum_{1 \leq t \leq t^*} x_{i^* t}$. Then, we compute the number of jobs which must be processed before $j$ in the interval $(d_{t^*-1}, d^*_t]$. We can assume that this is

$$q = \sum_{1 \leq i < i^*} x_{it^*} + (r - 1 - \sum_{1 \leq t < t^*} x_{i^* t})$$

(i.e., the number of jobs of type $i < i^*$ processed in the interval $t^*$, plus the number of jobs of type $i^*$ processed before $j$ but not already processed in a previous interval). Finally, the starting time of $j$ is given by $d_{t^*-1} + q$. Clearly, this procedure yields $S_J(j)$ in (strongly) polynomial time. Similar arguments show that $S_T$ can be computed pointwise in polynomial time. □

**Example 2 (Total deviation JIT).** An instance of this problem has the form $D = (s, n_1, n_2, \ldots, n_s)$, with the usual interpretation. All jobs have unit processing time. Assume that all jobs have been sequenced on a single machine, and let $x_{it}$ denote the number of jobs of type $i$ which have been sequenced in the interval $[0, t]$ $(i = 1, 2, \ldots, s; t = 1, 2, \ldots, n)$. The total weighted deviation JIT problem asks for a sequence which minimizes the total weighed deviation

$$\sum_{i=1}^{s} \sum_{t=1}^{n} F(x_{it} - t \frac{n_i}{n}), \tag{1}$$

where $F$ is a unimodal, convex function which penalizes the deviation between the actual cumulated production $x_{it}$ and the ideal production $tn_i/n$ up to time $t$.

Kubiak and Sethi (1991) and (1994) gave a polynomial total time list-generating algorithm with complexity $O(n^3)$ for this problem, by reformulating it as an assignment

problem. It is unknown whether its recognition or its evaluation versions can be solved in polynomial time, or even whether they are in $NP$ or co-$NP$. □

**Example 3 (Maximum deviation JIT).** This problem is similar to the previous one, except that the objective function (1) is replaced by a function penalizing the maximum deviation, namely

$$\max_{1\leq i\leq s} \max_{1\leq t\leq n} |x_{it} - t\,\frac{n_i}{n}|. \tag{2}$$

Brauner and Crama (2000) showed that the recognition version of the maximum deviation JIT problem, i.e. $\mathcal{SP}_1$, is in co-$NP$, but the exact complexity of $\mathcal{SP}_1$ is currently unknown. Steiner and Yeomans (1993) gave a polynomial total time list-generating algorithm for this problem. Interestingly, when the optimal objective value is known, then their algorithm produces the optimal schedule with polynomial delay (nothing similar seems to be known for the total deviation JIT problem, for instance).

Brauner and Crama (2000) proved that the evaluation version $\mathcal{SP}_2$ of the maximum deviation JIT problem can be solved in polynomial time when $s$ is fixed. In view of the previous remark, this also implies that the optimization version $\mathcal{SP}_3$ can be solved with polynomial delay when $s$ is fixed. But even in this case, we do not know whether there is a polynomial pointwise algorithm for computing $S_J$ or $S_T$. □

# 6 General scheduling problems

In this section, we propose an extension of the above discussion which encompasses more general scheduling problems involving multiple machines and job preemptions. We first have to agree on the definition of a schedule in this framework. Several options exist, but the following one seems quite generic. For an instance involving $n$ jobs and $m$ machines, we define a schedule to be a (finite) subset $S$ of $\{1,2,\ldots,n\}\times\{1,2,\ldots,m\}\times\mathbb{R}\times\mathbb{R}$. The interpretation 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 $t_2$. So, the elements of $S$ completely describe the Gantt chart of the schedule.

As in Section 4.1, a list-generating algorithm $\mathcal{A}$ for problem $\mathcal{SP}_3$ successively outputs the elements of an optimal schedule $S$. We denote by $\tau(l)$ the running time of $\mathcal{A}$ until it outputs the $l$-th element of $S$. In particular, $\tau(|S|)$ represents the running time of $\mathcal{A}$. All the complexity classes introduced in Section 4.1 can be generalized in a straightforward and consistent way: simply substitute $|S|$ for $n$ in all definitions. Proposition 1 can also be partially generalized as follows.

**Proposition 6** *If $\mathcal{A}$ is a list-generating algorithm for the optimization version $\mathcal{SP}_3$ of a general scheduling problem, then:*

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

**Proof** The same arguments apply as for Proposition 1. □

**Remark 4.** For preemptive problems, the size of an optimal schedule $S$ is not easily determined as a function of the input parameters. It may even happen that several optimal schedules exist, where some optimal schedules would be exponentially larger than others. In such a case, it may seem desirable to require that the algorithm $\mathcal{A}$ generate a schedule whose size is polynomially bounded in the size of the smallest optimal schedule. (Note that this difficulty is not specific to high multiplicity problems, but may even arise for traditional "low-multiplicity" problems.)

For many preemptive scheduling problems, it can be shown that there exists an optimal schedule $S$ involving a "small number" of preemptions, meaning typically that $|S| = O(nm)$. When this is the case, it it reasonable to require that $\mathcal{S}$ output a schedule of size polynomial in $nm$. □

**Remark 5.** For general scheduling problems, contrary to the single-machine non-preemptive case, we cannot claim that polynomial total time algorithms also run in pseudo-polynomial time. This is because the size of the optimal schedule $|S|$, as opposed to the number of jobs $n$, may not be bounded by a polynomial in $M$, viz. the largest number occurring in the instance (cf. the proof of Proposition 1). If the size of the schedule generated by the algorithm is polynomial in $nm$ (cf. previous remark), then polynomial total time implies pseudo-polynomial time, as in Proposition 1. □

As in Section 3, our definition of a schedule allows for the derivation of several associated mappings, which can be viewed as providing alternative readings of the schedule. For instance, we can define:

- a *job-oriented* description of the schedule, corresponding to the mapping

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

(this is a natural extension of the definition given in Section 3, which describes the complete routing of a job through the shop);

- a *machine-oriented* description of the schedule, corresponding to the mapping

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

(this gives the Gantt chart for a particular machine);

- a (*machine,time*)-*oriented* description of the schedule, corresponding to the mapping

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

($S_{MT}(k, t)$ describes the state of machine $k$ at time $t$, or at the first moment when the machine is busy after time $t$).

Here again, many other partial descriptions of the schedule could be thought of.

Extending our previous definitions, we can say that an algorithm $\mathcal{A}$ is a polynomial pointwise algorithm for $S_{MT}$ if, given any machine $k$ and time $t$, $\mathcal{A}$ outputs $S_{MT}(k, t)$ in polynomial time.

However, analyzing the complexity of the derived mappings $S_J$ or $S_M$ deserves more care, since these mappings are set-valued, rather than single-valued as in Section 3. In their case, it seems natural to combine 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.

# 7 Applications: general case

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 4 (Makespan minimization).** Consider first $P/pmtn/C_{max}$, i.e. the makespan minimization problem for a set of jobs to be processed preemptively over parallel identical machines. An instance of the problem is a vector

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

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

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

$$C^*_{max} = \max\{(\sum_{i=1}^{s} n_i p_i)/m, p_1, p_2, \ldots, p_s\},$$

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), 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 $EXP \setminus P$. 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 5 (Sum of completion times).** Consider the problem $Q2/pmtn/\sum_j C_j$. 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 $i = 1, 2, \ldots, s$,

$$\sigma_i(j) \;=\; \frac{p_i}{v_1 + v_2} - \left(\frac{p_i}{v_1 + v_2} - \sigma_i(0)\right)\left(-\frac{v_2}{v_1}\right)^j, \tag{3}$$

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 $j^*$. As in Example 1, 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 $t_1 = \sum_{1 \leq i < i^*} \sum_{1 \leq j \leq n_i} \sigma_i(j) + \sum_{1 \leq j < r} \sigma_{i^*}(j)$. 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. □

The results concerning the different models discussed in this section, and in Section 5, 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).

| | $\mathcal{SP}_1$ | $\mathcal{SP}_2$ | $\mathcal{SP}_3$ | |
|---|---|---|---|---|
| | | | 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 |

Table 1: Complexity of various problems

# 8 Discussion

The complexity of high multiplicity scheduling problems has been discussed by several authors, but it seems that a fully satisfactory framework has been missing so far for this discussion. The aim of this note is to propose such a framework.

A main (albeit obvious) observation is that the complexity of the task "output an optimal schedule" cannot be meaningfully discussed unless we explicitly clarify the form of the output. It makes a big difference, for instance, whether we want to generate all elements of a schedule, viewed as a set, or whether we just want to compute some elements of the schedule, viewed as a mapping.

In the classification scheme that we propose, we see that there is in fact no essential difference between the nature of the simple, compact encoding of an optimal schedule for $P/pmtn/C_{max}$ provided by the description of McNaughton's algorithm (1959), and the nature of the compact encoding of an optimal schedule for $1/p_j = 1/\sum_j w_j U_j$ provided by the solution of the transportation problem in Hochbaum and Shamir (1991) (cf. Table 1). In both cases, what comes out of the algorithm and of its proof of correctness is not an *explicit* description of the optimal schedule, but an *implicit* description – an encoding – which can be used either to generate the schedule in extension, or to compute various derived mappings in a pointwise fashion (in a 2-phase approach).

Different encodings of the optimal schedule, however, may differ in the extent to which they allow an efficient decoding into explicit schedules. The classification of algorithms proposed in this paper provides one way of distinguishing algorithms on this basis.

Algorithms for high multiplicity scheduling problems may also be compared on other grounds than those discussed in previous sections. In particular, it may be desirable

to classify them on the basis of their space complexity, rather than time complexity only. For instance, Kubiak and Sethi's (Kubiak and Sethi (1991) and Kubiak and Sethi (1994)) formulation of the total deviation just-in-time problem requires $\Omega(n)$ time *and* space. Similar issues have been discussed, in various contexts, by Lawler, Lenstra, and Rinnooy Kan (1980), Dyer (1983) and Johnson, Yannakakis, and Papadimitriou (1988).

Finally, 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 6. It would be useful to investigate some of these relations in future work.

# References


Bar-Noy, A., R. Bhatia, J. S. Naor, and B. Schiber (1998). Minimizing service and operation costs of periodic scheduling. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 11–20.

Brauner, N. and Y. Crama (2000). Facts and questions about the maximum deviation just-in-time scheduling problem. Working paper GEMME 0013, University of Liège, Liège, Belgium. Revised May 2001.

Brauner, N., G. Finke, and W. Kubiak (2000). Complexity of one-cycle robotic flow-shops. Working paper GEMME 0001, University of Liège, Liège, Belgium.

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*. San Francisco: W.H. Freeman and Company.

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

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.

Hurink, J. and S. Knust (1998). Flow shop problems with transportation times and a single robot. Osnabrücker Schriften zur Mathematik 201, Universität Osnabrück, Osnabrück, Germany.

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 (1993). Polynomial algorithms for multiprocessor scheduling problems with a small number of job lengths. Faculty of Commerce Working Paper 93-MSC-008, University of British Columbia, Vancouver, B.C., Canada.

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*. Englewood Cliffs, N.J.: Prentice Hall.

Pinedo, M. (1995). *Scheduling: Theory, Algorithms and Systems*. Englewood Cliffs, N.J.: Prentice Hall.

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

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.