A framework for the complexity of high-multiplicity scheduling problems

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

Revised - February 2005

[*]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

Abstract

The purpose of this note is to propose a complexity framework for the analysis of high multiplicity scheduling problems. Part of this framework relies on earlier work aiming at the definition of output-sensitive complexity measures for the analysis of algorithms which produce "large" outputs. However, different classes emerge according as we look at schedules as sets of starting times, or as related single-valued mappings.

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

1 Introduction

The purpose of this note is to propose a complexity framework for the analysis of so-called *high multiplicity scheduling (HMS) problems.* Such problems have been 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), (Hochbaum and Shamir 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 HMS problems into a more precise computational complexity framework. As usual, this requires a complexity study of the algorithms that solve the problems. It will become clear quickly, however, that the meaning of the task "to solve a HMS problem" is not entirely obvious. Therefore we propose a framework that allows to study a variety of solution representations and algorithms for HMS problems. The desired classification of HMS *problems* will follow naturally from this classification of HMS *algorithms*, just like the definition of polynomially solvable decision problems, for instance, follows from the definition of polynomial algorithms for such problems. Although the paper does not contain any deep theoretical results, we believe that such classification may prove useful for a precise analysis of HMS problems and a precise statement of algorithmic performance.

For the sake of clarity, we restrict ourselves to non-preemptive one-machine scheduling problems. More complex problem formulations are tackled by (Brauner, Crama, Grigoriev, and van de Klundert 2001) or (Grigoriev 2003).

2 High multiplicity scheduling problems

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 $s \ll n$, 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 $\mathcal{SP}$ consists of the following data:

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

This kind of situation is encountered in repetitive manufacturing environments (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, Shamir, and Shanthikumar 1992).

Consider now a generic instance $D = (s, n_1, n_2, \ldots, n_s, \Delta)$ of a (one-machine non-preemptive) high multiplicity scheduling problem $\mathcal{SP}$, where Δ comprises all the relevant job attributes. We assume that all the entries of D are integral, and we denote by $n = \sum_{1 \le i \le s} n_i$ the number of jobs to be processed. We also assume without loss of generality 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.

The input size of instance D is $|D| = O(\sum_{1 \le i \le s} \log n_i + sL) = O(s \log n + sL)$, where L is again the largest input size of an attribute value. Typically, this input size is much smaller than $O(nL)$, as is e.g. the case when s is viewed as a constant and therefore $|D| = O(\log n + L) \ll O(nL)$. 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 $\mathcal{SP}$. As observed by (Hochbaum and Shamir 1990), (Hochbaum and Shamir 1991), (Hochbaum, Shamir, and Shanthikumar 1992), (Clifford and Posner 2001), 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. In order to develop this point more completely, we need to introduce more terminology and notations.

A schedule for the instance D can be viewed as an assignment $S\colon \{1, 2, \ldots, n\} \rightarrow \mathbb{R}$ where $S(j)$ denotes the starting time of job j $(j = 1, 2, \ldots, n)$, and where S may be (and usually is) restricted to belong to a set $\mathcal{F}_D$ of *feasible* schedules associated with D.

We let $f_D\colon \mathcal{F}_D \rightarrow \mathbb{R}$ be the objective function to be minimized over $\mathcal{F}_D$. 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 values $S(1), S(2), \ldots, S(n)$), $f_D(S)$ can be computed in time polynomial in $|D|$ and n.

As in (Papadimitriou and Steiglitz 1982), 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). Hence membership in NP is a non trivial issue for many high multiplicity scheduling problems. Similar problems pop up when considering membership in P.

In spite of these difficulties, many high multiplicity scheduling problems have been proved to be polynomially solvable (see for instance (Brauner, Finke, and Kubiak 2003), (Clifford and Posner 2000), (Clifford and Posner 2001), (Granot and Skorin-Kapov 1993), (Hochbaum and Shamir 1990), (Hochbaum and Shamir 1991), (Hochbaum, Shamir, and Shanthikumar 1992), (Hurink and Knust 2001), (McCormick, Smallwood, and Spieksma 2001), (Munier and Sourd 2003), etc.) or in co-NP ((Brauner and Crama 2004)) or NP-hard ((Clifford and Posner 2000), (Clifford and Posner 2001), (Posner 1985), (Bar-Noy, Bhatia, Naor, and Schiber 2002), 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). 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 the 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 1991) 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

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

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). $\square$

Let us now turn to the optimization problem $\mathcal{SP}_3$. Few authors have attempted to discuss precisely what it means to "solve" $\mathcal{SP}_3$. (Hochbaum and Shamir 1991) and (Hochbaum, Shamir, and Shanthikumar 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 the above example, 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 4). In the next section, we propose more general models for describing a solution of problem $\mathcal{SP}_3$, which allow us to obtain a more precise complexity classification.

3 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 focus on the *set of starting-times*, or on the *computation of the mapping* S.

3.1 List-generating algorithms

In our first interpretation, we assume that the set $\{S(1), S(2), \ldots, S(n)\}$ is to be generated in extension: this may be the most natural interpretation of the requirement "output an optimal schedule S".

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 values $(\pi^1, S(\pi^1))$, $(\pi^2, S(\pi^2))$, $\ldots$, $(\pi^n, S(\pi^n))$, where S is an optimal schedule and $(\pi^1, \pi^2, \ldots, \pi^n)$ is a permutation of the job-set.

Examples of list-generating algorithms are found in (Brauner and Crama

2004), (Kubiak and Sethi 1991), (Kubiak and Sethi 1994), (Munier and Sourd 2003), (Steiner and Yeomans 1993), etc. 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. $(\pi^1, S(\pi^1)), (\pi^2, S(\pi^2))$, … , $(\pi^j, S(\pi^j))$. 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:

- *polynomial total time* if $\tau(n)$ is polynomially bounded in n and $|D|$;
- *polynomial incremental 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 are motivated by the same considerations as the definitions in (Johnson, Yannakakis, and Papadimitriou 1988). Let us discuss them briefly.

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 the starting times of all n jobs must grow at least linearly with n.

Polynomial incremental 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 π^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 $|D|$.

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, Yannakakis, and Papadimitriou 1988). We also feel that it is one of the most meaningful requirements that may apply to algorithms for HMS problems. Indeed, in contrast, polynomial time is the usual concept from complexity theory and is only mentioned here for the sake of completeness: if $\mathcal{SP}_3$ can be solved in polynomial time, then n must be bounded by a polynomial in $|D|$ for all instances of this problem (since $\tau(n)$ is at least linear in n), and the problem does not qualify as a high multiplicity problem.

Clearly, we can define complexity classes for HMS problems based on the corresponding classification of algorithms: we say that problem $\mathcal{SP}$ is solvable in polynomial total time (resp., incremental time, or delay) if there is an algorithm for $\mathcal{SP}_3$ with the corresponding running time.

The following relationships hold:

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:*

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

Proof All the implications are easy. For instance, if 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 polynomially bounded in n and $|D|$. Hence, A runs in polynomial total time. □

3.2 Pointwise algorithms

In this 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 the mapping S *pointwise*.

Definition 3. A *pointwise algorithm* for $\mathcal{SP}_3$ is an algorithm $\mathcal{A}$ such that (1) on the input (D, j), algorithm $\mathcal{A}$ outputs $S(j)$ $(j = 1, 2, \ldots, n)$, and (2) $\{\, S(j) \,:\, j \in \{1, 2, \ldots, n\}\,\}$ defines an optimal schedule for D.

Since a pointwise algorithm produces a single numerical output for each input string, the classical complexity measures apply to it without modification. In particular, the existence of a polynomial pointwise algorithm for $\mathcal{SP}_3$ simply means that the optimal schedule can be queried in polynomial time or, in other words, that the function $S : \{1, 2, \ldots, n\} \to \mathbb{R}$ 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 algorithm;*

(b) *if $\mathcal{SP}_3$ has a polynomial pointwise algorithm, then $\mathcal{SP}_3$ has a polynomial-delay list-generating algorithm.*

Proof Statement (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.

For statement (b), observe that a polynomial pointwise algorithm can be called n times to compute successively $S(1), S(2), \ldots, S(n)$. Since the running time of each call is polynomial in $|D|$, the resulting list-generating algorithm runs with polynomial delay. $\square$

So, intuitively, polynomial pointwise algorithms fall somewhere between polynomial and polynomial delay list-generating algorithms in the hierarchy described in Proposition 1. We already mentioned in Section 3.1 that genuine HMS problems do not have polynomial time list-generating algorithms; thus, statement (a) is rather vacuous from that point of view. Note, however, that its converse does not hold in general, meaning that certain HMS problems do have polynomial pointwise algorithms. Example 1 in Section 4 will illustrate this point.

On the other hand, we conjecture that the converse of statement (b) also fails, but we cannot establish this conjecture (see Example 3 in Section 4).

Interestingly, many known examples of polynomial pointwise algorithms actually consist of two distinct algorithms: a first algorithm $\mathcal{A}_e$ which solves $\mathcal{SP}_2$ while producing a compact "encoding" ("certificate") Σ of the optimal schedule S, and a second algorithm $\mathcal{A}_u$ which computes S by "decoding" the output produced by $\mathcal{A}_e$. Let us formulate these notions in more precise terms.

Definition 4. A 2-*phase 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 Σ;

(2) on the input (D, j, Σ), $\mathcal{A}_u$ outputs $S(j)$ $(j = 1, 2, \ldots, n)$, and

(3) $\{\, S(j) \,:\, j \in \{1, 2, \ldots, n\}\,\}$ defines an optimal schedule for D.

The string Σ in this definition represents the encoding of the optimal solution. It can be viewed, in a sense, as a natural counterpart of the compact encoding of the input.

We say that a 2-phase 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. In particular, when this is the case, the size of Σ must be polynomially related to the size of D. For examples, see e.g. (Clifford and Posner 2000), (Clifford and Posner 2001), (Granot and Skorin-Kapov 1993), (Hochbaum and Shamir 1990), (Hochbaum and Shamir 1991), (Hochbaum, Shamir, and Shanthikumar 1992), (Hurink and Knust 2001), (McCormick, Smallwood, and Spieksma 2001).

The definition of pointwise algorithms may appear to be slightly less restrictive than the definition of 2-phase algorithms. But in fact, the following equivalence holds.

Proposition 3 *Problem $\mathcal{SP}_3$ has a polynomial pointwise algorithm if and only if it has a polynomial 2-phase algorithm.*

Proof Assume that $\mathcal{SP}_3$ has a polynomial pointwise algorithm $\mathcal{A}$. We define $\mathcal{A}_e$ as the algorithm which always returns the empty string ϕ. The decoding algorithm $\mathcal{A}_u$ can be identified with $\mathcal{A}$: when running on the input (D, j, ϕ), the algorithm simply ignores the empty string.

Conversely, if $\mathcal{SP}_3$ has a polynomial 2-phase algorithm $(\mathcal{A}_e, \mathcal{A}_u)$, then a polynomial pointwise algorithm $\mathcal{A}$ can be defined as follows. When handed the input (D, j) $(j = 1, 2, \ldots, n)$, the algorithm $\mathcal{A}$ first runs $\mathcal{A}_e$ on D to obtain Σ, then it runs $\mathcal{A}_u$ on (D, j, Σ) to compute $S(j)$. Note that the total running time of this procedure is polynomial in $|D|$. $\square$

Thus, polynomial pointwise and 2-phase algorithms turn out to be equivalent.

4 Applications

Example 1 (Weighted number of tardy jobs – continued). A discussion of this problem was already started in Section 2. Let us now show that the approach in (Hochbaum and Shamir 1991) leads to a polynomial pointwise algorithm for the optimization version of this problem. This algorithm is best viewed as a two-phase approach. Indeed, the solution (x_{it}) of the transportation problem can be computed in $O(s \log s)$ and constitutes the required encoding Σ. Then, given a job index j, we first determine the type of this job, i.e. the unique index i^* such that $\sum_{1 \le i < i^*} n_i < j \le \sum_{1 \le i \le i^*} n_i$. If $r = j - \sum_{1 \le i < i^*} n_i$, 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 $\sum_{1 \le t < t^*} x_{i^*t} < r \le \sum_{1 \le t \le t^*} x_{i^*t}$. 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

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

(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 $S(j)$ in (strongly) polynomial time. $\square$

Example 2 (Total deviation JIT). An instance of this problem is $D = (s, n_1, n_2, \ldots, n_s)$, 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 deviation JIT problem asks for a sequence which minimizes the total "weighted" 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 (Kubiak and Sethi 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 version can be solved in polynomial time, or even whether it is in NP or co-NP. $\square$

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

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

(Brauner and Crama 2004) 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 hold for the total deviation JIT problem, for instance).

(Brauner and Crama 2004) proved that the evaluation version $\mathcal{SP}_2$ 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 the optimal schedule. $\square$

5 Discussion

The complexity of high multiplicity algorithms 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 trivial) observation is that the complexity of the task "output an optimal schedule" cannot be meaningfully discussed unless we explicitly clarify the form of its 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 addition, different "compact" encodings of the optimal schedule may differ in the extent to which they allow an efficient decoding into explicit schedules. The complexity classification proposed in this paper provides one way of distinguishing algorithms, and hence problems, on this basis. The results obtained by various authors suggest that the relationship between different complexity classes may go deeper than the simple implications mentioned in Proposition 1.

Extensions of this framework to more complex scheduling problems (involving multiple machines and job preemptions) are proposed in (Brauner, Crama, Grigoriev, and van de Klundert 2001) and (Grigoriev 2003).

6 Acknowledgements

The authors are grateful to Tom McCormick, Maurice Queyranne, Gerhard Woeginger and two anonymous referees for their stimulating comments.

This research 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 acknowledges support from the EU project APPOL (IST-1999-14084).

References

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

N. Brauner and Y. Crama, "The maximum deviation just-in-time scheduling problem", Discrete Applied Mathematics vol.134, pp. 25-50, 2004.

N. Brauner, Y. Crama, A. Grigoriev and J. van de Klundert, "On the complexity of high-multiplicity scheduling problems", University of Liège, Liège, Belgium, Working paper GEMME 0110, 2001.

N. Brauner, G. Finke and W. Kubiak, "Complexity of one-cycle robotic flow-shops", Journal of Scheduling vol.6, pp. 355-371, 2003.

J.J. Clifford and M. E. Posner, "High multiplicity in earliness-tardiness scheduling", Operations Research vol.48, pp.788-800, 2000.

J.J. Clifford and M.E. Posner, "Parallel machine scheduling with high multiplicity", Mathematical Programming vol.89, pp.359-383, 2001.

S.S. Cosmadakis and C.H. Papadimitriou, "The traveling salesman problem with many visits to few cities", SIAM Journal on Computing vol.13, pp.99-108, 1984.

M.E. Dyer, "The complexity of vertex enumeration methods", Mathematics of Operations Research vol.8, pp.381-402, 1983.

M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman: San Francisco, CA, 1979.

F. Granot and J. Skorin-Kapov, "On polynomial solvability of the high multiplicity total weighted tardiness problem", Discrete Applied Mathematics vol.41, pp.139-146, 1993.

A. Grigoriev, "High Multiplicity Scheduling Problems", Maastricht University, The Netherlands, Doctoral thesis, 2003.

D.S. Hochbaum and R. Shamir, "Minimizing the number of tardy job units under release time constraints", Discrete Applied Mathematics vol.28, pp.45-57, 1990.

D. S. Hochbaum and R. Shamir, "Strongly polynomial algorithms for the high multiplicity scheduling problem", Operations Research vol.39, pp.648-653, 1991.

D.S. Hochbaum, R. Shamir and J.G. Shanthikumar, "A polynomial algorithm for an integer quadratic nonseparable transportation problem", Mathematical Programming vol.55, pp.359-376, 1992.

J. Hurink and S. Knust, "Makespan minimization for flow-shop problems with transportation times and a single robot", Discrete Applied Mathematics vol. 112, pp. 199-216, 2001.

D.S. Johnson, M. Yannakakis and C.H. Papadimitriou, "On generating all maximal independent sets", Information Processing Letters vol.27, pp.119-123, 1988.

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

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

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

S.T. McCormick, S.R. Smallwood and F.C.R. Spieksma, "A polynomial algorithm for multiprocessor scheduling with two job lengths", Mathematics of Operations Research, vol. 26, pp. 31-49, 2001.

J. Miltenburg, "Level schedules for mixed-model assembly lines in just-in-time production systems", Management Science vol.35, pp.192-207, 1989.

A. Munier and F. Sourd, "Scheduling chains on a single machine with non-negative time-lags", Mathematical Methods of Operations Research vol.57, pp.111-123, 2003.

C.H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice Hall: Englewood Cliffs, N.J., 1982.

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

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

H.N. Psaraftis, "A dynamic programming approach for sequencing groups of identical jobs", Operations Research vol.28, pp.1347-1359, 1980.

M. Rothkopf, "The travelling salesman problem: On the reduction of certain large problems to smaller ones", Operations Research vol.14, pp.532-533, 1966.

G. Steiner and J. S. Yeomans, "Level schedules for mixed-model, just-in-time processes", Management Science vol.39, pp.728-735, 1993.