High Multiplicity Scheduling on One Machine with Forbidden Start and Completion Times

Michaël Gabay, Christophe Rapine, Nadia Brauner

High Multiplicity Scheduling on One Machine with Forbidden Start and Completion Times

Michaël Gabay*, Christophe Rapine†, Nadia Brauner*

Abstract

We are interested in a single machine scheduling problem where no task can either start or end on some dates and the input is given using a compact encoding. The aim is to minimize the makespan. We present a polynomial time algorithm for large diversity instances (when the number of different processing times is bigger than the number of forbidden dates). We also show that this problem is fixed parameter tracktable when the number of forbidden dates is fixed, regardless of tasks characteristics.

Keywords: Scheduling, High Multiplicity, Availability Constraints, Parametrized Complexity

1 Introduction

We consider the single machine scheduling problem with forbidden start and end instants. In this problem, we have several tasks, described by their processing times and some forbidden instants. On those instants, no task can start, nor end. We denote by N the set of jobs indices $\{1, \ldots, n\}$ and by $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$ the set of forbidden instants. The aim is to find a schedule minimizing the makespan. We suppose that the input is encoded using a compact encoding, as described in the next section. Using Graham notations, the problem is denoted by $1|\text{Fse}|C_{\max}$. In the following, we will use the same notations and definition as in [16].

We disregard trivial infeasible instances, that is, when 0 or $\sum_i p_i$ is forbidden.

2 High Multiplicity

The term *High Multiplicity* (HM for short) was introduced by Hochbaum and Shamir [9] to refer to a compact encoding of instances where identical jobs appear many times. Compared to a traditional encoding where each job is described, in a HM encoding each *type* is described only once, along with its multiplicity (the number of jobs of this type). Thus the size of a HM encoding depends linearly on the number of types but only logarithmically on the number of jobs. As a consequence, a polynomial time algorithm under the standard encoding may become exponential under a HM encoding of the instances, which

*Grenoble-INP / UJF-Grenoble 1 / CNRS, G-SCOP UMR5272 Grenoble, F-38031, France
†Université de Lorraine, Laboratoire LGIPM, Ile du Saulcy, Metz, F-57045, France

is the case of our algorithms. HM scheduling and more generally HM combinatorial optimization has become an active domain in recent years [3, 6, 7].

3 A polynomial time algorithm for large diversity instances

Let $x = (N, \mathcal{F})$ be a large diversity instance. We denote for short by $q = |\langle N \rangle|$ the number of types. Indexing types by decreasing order of their processing times, set N is represented in HM encoding by its multiplicity vector $(n_1, \ldots, n_q)$ together with the processing vector $(p_{\langle 1 \rangle}, \ldots, p_{\langle q \rangle})$, where n_i and $p_{\langle i \rangle}$ are the number of jobs of the ith type and its processing time, respectively. The size $|x|$ of the instance under a HM encoding is thus in $\Omega(q(\log n + \log p_{\langle 1 \rangle}) + k \log \gamma_k)$. Hence $|x|$ can be in $\mathcal{O}(q \log n)$ while algorithm L-partition runs in $\mathcal{O}(k^3 n)$, which can be exponential with respect to $|x|$. Note that an idle-free schedule π is simply a permutation of the jobs. In HM scheduling, it may be not obvious to determine if there exists (optimal) schedules with a compact encoding, *i.e.* polynomial in $|x|$. For $1|\text{FSE}|C_{\max}$ it is readily that the schedule of the jobs between two forbidden instants is meaningless, and thus the jobs of the same type can be scheduled consecutively. As a consequence any idle-free schedule has a polynomial encoding as a sequence of couples (i_l, α_l), where i_l designates a type and α_l the number of jobs of this type scheduled consecutively. We use this representation inside this section.

To achieve a polynomial time algorithm, we need two ingredients. Firstly, we can not afford to allocate only one job at a time. Secondly, we have to design a more efficient approach than L-partition. To cope with the latter point, consider a partial schedule π, completing at time t. We say that π is an *optimal prefix* if there exists an optimal schedule of the form $\pi\sigma$. In this situation, the problem reduces to finding an optimal schedule starting at time t on the remaining set N' of jobs. Notice that algorithm L-partition finds an optimal suffix, with the drawback of computing a potentially long sequence of valid partitions to obtain it. Now how can we assert that a partial schedule π is an optimal prefix? The response is quite simple: due to Theorem 2 from [16], a sufficient condition is that π is idle-free, and that the remaining instance $(N', \mathcal{F}' = \mathcal{F} \cap [t, +\infty])$ is a large diversity instance. Based on these ideas, we derive Algorithm 1.

We give here a comprehensive description of this algorithm. The basic idea is to reduce the instance to have only one job of each type and only $|\mathcal{F}| + 1$ types, such that L-partition algorithm can be used efficiently. Initially one job of each of the $|\mathcal{F}| + 1$ largest types is put aside in order to control the number of types. Let F be this set, and let us call *additional* jobs the set $N \backslash F$. We schedule then iteratively all the additional jobs of type 1, then all the additional jobs of type 2, ..., as long as they all fit before the first forbidden instant γ_1. When this process terminates, either only set F remains to schedule, or there is not enough room left before γ_1 to schedule all the additional jobs of the ith type. In the latter case the algorithm schedules as much as possible of jobs of type i before γ_1, and tries to cross forbidden instant γ_1. Our second idea is here to ensure that the schedule of each job of F permits to cross at least one forbidden instant in order to keep a large diversity instance. We claim that Algorithm 1 is correct, *i.e.* delivers an optimal prefix π. In addition if $(N', \mathcal{F}')$

Algorithm 1 Optimal Prefix Algorithm

Require: a large diversity instance $(N,\mathcal{F})$ with types indexed in decreasing order.

Ensure: an optimal prefix π

```
set m_i = n_i − 1 if i ≤ |F| + 1, m_i = n_i otherwise
set i = 1 ; t = 0 ; π = ∅
while i ≤ |⟨N⟩| and t + m_i p_⟨i⟩ < γ_1 do
  // Append to π all the m_i jobs of type i
  π = π(i, m_i) ; t = t + m_i p_⟨i⟩ ; i = i + 1
end while
if i > |⟨N⟩| then
  return π // Only |F| + 1 jobs remains to schedule
end if
// Append as many as possible jobs of type i before γ_1
α = ⌈(γ_1 − t)/p_⟨i⟩⌉ − 1 ; π = π(i, α) ; t = t + α p_⟨i⟩ ;
// Extend π to complete after time γ_1
for all l ∈ 1, ..., |F| + 1 such that t + p_⟨l⟩ ≥ γ_1 do
  if t + p_⟨l⟩ ∉ F then
    return π(l, 1)
  end if
end for
for all l ∈ 2, ..., |F| + 1 such that t + p_⟨l⟩ < γ_1 do
  if t + p_⟨l⟩ + p_⟨1⟩ ∉ F then
    return π(l, 1)(1, 1)
  end if
end for
```

is the remaining instance to schedule, then $(N', \mathcal{F}')$ is a large diversity instance and:

1. either $|\mathcal{F}'| < |\mathcal{F}|$, *i.e.* we have strictly less forbidden instants,
2. or $|N'| = |\langle N' \rangle| = |\mathcal{F}| + 1$, *i.e.* all the remaining jobs have distinct processing times.

In the first case, we recursively call the prefix algorithm on instance $(N', \mathcal{F}')$. The second case corresponds to the basis of the recursion: we simply solve instance $(N', \mathcal{F}')$ using the L-partition algorithm. Since N' contains at most $(k+1)$ jobs, the running time of L-partition algorithm on this instance is in $\mathcal{O}(k^4)$. We prove our claim in the following theorem:

Theorem 1 *Problem* $1|\text{FSE}|C_{\max}$ *is polynomial under HM encoding for large diversity instances, and can be solved in time* $\mathcal{O}(k|\langle N \rangle| + k^4)$

Proof. Correctness of Algorithm 1. Let $(N', \mathcal{F}')$ be the instance remaining to schedule at the end of Algorithm 1. Recall that F denotes a set with exactly one job of the $|\mathcal{F}| + 1$ largest types of N. Let $A = N \backslash Z$ be the additional jobs. If only set F remains to schedule at the end of the algorithm, we are clearly in the second case of our claim. Otherwise the algorithm has stopped the first loop on a type i such that all its additional jobs cannot be scheduled before γ_1. At this point, there remains at least one unscheduled job of type i in A, and possibly another in F, if $i \leq k+1$. Let $t < \gamma_1$ be the current completion time of the schedule, and consider the partition $F = \mathcal{S} \cup \mathcal{L}$ defined by $\mathcal{L} = \{j \in F \mid t + p_j \geq \gamma_1\}$. Notice that $\mathcal{L}$ is not empty as $t + p_{\langle i \rangle} \geq \gamma_1$; in particular a job of type 1 belongs to $\mathcal{L}$. By construction Algorithm 1 tries to extend π to complete after the first forbidden instant γ_1, which corresponds to the first case of our claim. We have to prove that it will always succeed, and that $(N', \mathcal{F}')$ is a large diversity instance. Basically we show in the following that if π completes after the lth forbidden instant, at most l jobs of F have been scheduled in π. Indeed we then have $|\langle N' \rangle| \geq |F| - l > |\mathcal{F}| - l$ and $|\mathcal{F}'| \leq |\mathcal{F}| - l$, which ensures that $(N', \mathcal{F}')$ is a large diversity instance. Consider the last two loops of the algorithm. If one job of $\mathcal{L}$ can be scheduled, the property clearly holds as π completes after time γ_1. If this is not possible, instant $t + p_j$ is forbidden for all jobs j of $\mathcal{L}$. By construction any job of $\mathcal{S}$ can be scheduled before time γ_1. Therefore a simple counting argument ensures that there exists a job $s \in \mathcal{S}$ that can be scheduled at time t immediately followed by a job of type 1. If $t + p_s + p_{\langle 1 \rangle} \geq \gamma_2$, *i.e.* π completes after time γ_2, we are done. Otherwise, we have $t + p_{\langle 1 \rangle} < \gamma_2$. In this case $|\mathcal{F}'| = |\mathcal{F}| - 1$, while we apparently use 2 jobs of F. However since instant $t + p_{\langle 1 \rangle}$ is forbidden by construction, in fact we have $t + p_{\langle 1 \rangle} = \gamma_1$ and as a consequence $i = 1$. As we noticed, there is at least one unscheduled job of type i in A, i.e. we can use an additional job of type $i = 1$ to schedule after s. Hence we have $|\langle N' \rangle| \geq |\langle N \rangle| - 1$ which completes the proof of correctness of the algorithm.

Time complexity. We use the classic convention that basic operations on integers (addition, division,...) are performed in constant time. Then the time complexity of Algorithm 1 is in $\mathcal{O}(k + q)$, which is in $\mathcal{O}(q)$ for large diversity instances. To solve Problem $1|\text{FSE}|C_{\max}$, we call Algorithm 1 on the set of unscheduled jobs as long as there is still some forbidden instants in the future or

that this set is not reduced to F. Thus we have at most k calls to Algorithm 1 as at least one forbidden instant is crossed each time, possibly followed by a call to L-partition algorithm on an instance containing at most $k+1$ jobs. Therefore the overall complexity is in $\mathcal{O}(k|\langle N\rangle| + k^4)$. □

Notice that Theorem 1 provides a better time complexity that the one based exclusively on L-partition algorithm, running in time $\mathcal{O}(k^3 n)$, even for a traditional encoding of the instances.

4 A polynomial algorithm with a fixed number of Fse

Theorem 2 $1|\text{Fse}|C_{\max}$ *is Fixed Parameter Tracktable for parameter* k*, even under HM encoding.*

The main idea is to use Algorithm 1 on large diversity instances (it performs in $\mathcal{O}(|\langle N\rangle|)$ time for fixed k) and to solve an integer program otherwise.

In order to prove that the integer program can be solved in polynomial time, we will use the following result from Eisenbrand [17]:

Theorem 3 (Eisenbrand, 2003) *An integer program of binary encoding length* s *in fixed dimension, which is defined by a fixed number of constraints, can be solved with* $\mathcal{O}(s)$ *arithmetic operations on rational numbers of binary encoding length* $\mathcal{O}(s)$*.*

Remark that for small diversity instances, we have $k \geq |\langle N\rangle|$. Hence, if we have a valid integer program whose dimension and number of constraints are bounded by a polynomial in k, then, using theorem 3, we have shown that $1|\text{Fse}|C_{\max}$ is FPT with parameter k. Moreover, remark that having one decision variable or one constraint for each task would make it impossible to bound the integer program dimension and/or number of constraints by a function of k. Hence, we have to use an HM encoding in the IP and therefore the result will be valid under HM encoding.

In the following, we will present the integer program. The main idea in this program is to use the fact that any large diversity instance admits an idle-free schedule. So, we model the problem as a large diversity instance by modeling idle times as a large enough number of optionnal jobs (we don't have to schedule them). Then, we minimize the makespan which is the first date where all real jobs have been processed.

Tasks

We consider 3 kinds of tasks

- Real jobs : processing time p_i, multiplicity m_i for $i = 1, \ldots, k$.
- $k + 1$ idle time jobs, ensures that there exists an idle free schedule: processing time $p_i = i - k$ for $i = k + 1, \ldots, 2k + 1$.

- A task that allows to skip all remainind FSE if the schedule finishes before the last one: processing time $p_{2k+2} = \gamma_k + 1$, multiplicity unbounded (but at most 1 will be used in an optimal solution).

Remark that the number of real jobs do not depend on $|\langle N \rangle|$. Actually, for $i = 1, \ldots, |\langle N \rangle|$, we set p_i and m_i using real jobs details and then, for $i = |\langle N \rangle| + 1, \ldots, k$ we set $p_i = m_i = 0$.

We will also need a big M value (denoted Q). An obvious choice would be $\gamma_k + \sum_{i=1}^{|\langle N \rangle|} m_i p_i$ but since it is shown in [16] that any list scheduling algorithm produces a schedule with makespan at most $2k + \sum_{i=1}^{|\langle N \rangle|} m_i p_i$, it is an upper bound on the optimal schedule makespan and we can set $Q = 2k + \sum_{i=1}^{|\langle N \rangle|} m_i p_i$.

4.1 Decision variables

We now describe the decision variables:

$$m_{ij} : \text{number of tasks of type } i \text{ completed by time } \gamma_j \tag{1}$$

$$S_{jf} = 1 \text{ iff a task covers exactly the instants } \gamma_j \text{ till } \gamma_{f-1} \text{ (included).} \tag{2}$$

$$x_{ij} = 1 \text{ iff a task of type } i \text{ covers the instant } \gamma_j \text{ and this task does not cover the previous FSE instant.} \tag{3}$$

$$y_j = 1 \text{ iff all real tasks have been completed by time } \gamma_j \tag{4}$$

$$C : \text{makespan of the schedule} \tag{5}$$

We will also denote by W_j the total work completed by time γ_j. This is a short-hand for $\sum_{i=1}^{(2k+2)} p_i m_{ij}$.

The total number of decision variable is given by $f(k)$ a function of k only – $f(k) = \mathcal{O}(k^2)$.

4.2 Objective

$$\min C$$

4.3 Constraints

- All FSE are covered (S_{jf} defines a $1 - (k+1)$ path)

$$\sum_{f=2}^{k+1} S_{1f} = 1 \tag{6}$$

$$\sum_{f=1}^{k} S_{f,k+1} = 1 \tag{7}$$

$$\sum_{j=1}^{f-1} S_{jf} = \sum_{l=f+1}^{k+1} S_{fl} \qquad \forall f = 2, \ldots, k \tag{8}$$

- All the work W_j must be completed by time γ_j :

$$W_j \leq \gamma_j - 1 \qquad \forall j = 1, \ldots, k+1 \tag{9}$$

- The amount of work completed can not increase between time γ_j and γ_{f-1} if $S_{jf} = 1$

$$W_l \leq W_j + Q(1 - \sum_{f=l+1}^{k+1} S_{jf}) \qquad \forall j < l \qquad (10)$$

- If $S_{jf} = 1$ and a task i covers γ_j, then $W_j + p_i$ should be in $[\gamma_{f-1}+1, \gamma_f - 1]$

$$W_j + \sum_{i=1}^{2k+2} p_i x_{ij} \geq \sum_{f=j+1}^{k+1} (\gamma_{f-1} + 1) S_{jf} \qquad \forall j = 1, \ldots, k \qquad (11)$$

$$W_j + \sum_{i=1}^{2k+2} p_i x_{ij} \leq \sum_{f=j+1}^{k+1} (\gamma_f - \gamma_j) S_{jf} + \gamma_j - 1 \qquad \forall j = 1, \ldots, k \qquad (12)$$

- A task covers γ_j as a first Fse instant if and only if $S_{jf} = 1$ for some index $f > j$:

$$\sum_{i=1}^{2k+2} x_{ij} = \sum_{f=j+1}^{k+1} S_{jf} \qquad \forall j = 1, \ldots, k \qquad (13)$$

- If a task i covers γ_j, then the number of tasks i processed should be updated by the end of i.

$$m_{i,f+1} \geq m_{if} \qquad \forall i = 1, \ldots, 2k+2, f = 1, \ldots, k \qquad (14)$$

$$m_{if} \geq m_{ij} + x_{ij} + S_{jf} - 1 \qquad \forall i,\ 1 \leq j < f \leq k+1 \qquad (15)$$

- Assign all real tasks

$$m_{i,k+1} = m_i \qquad \forall i = 1, \ldots, k \qquad (16)$$

- Set $y_j = 0$ if all the (real) tasks are not completed before time γ_j

$$\sum_{i=1}^{k} m_{ij} \geq \sum_{i=1}^{k} m_i y_j \qquad \forall j = 1, \ldots, k \qquad (17)$$

- The makespan should equal the first W_j such that $y_j = 1$

$$C \geq W_1 \qquad (18)$$

$$C \geq W_j - y_{j-1} Q \qquad \forall j = 2, \ldots, k+1 \qquad (19)$$

The total number of constraints is given by $g(k)$ a function of k only – $g(k) = \mathcal{O}(k^3)$.

4.4 Correctness

Proof. The IP is always feasible since we can schedule task $2k+2$ followed by all real tasks. Moreover, there is always an optimal idle-free schedule since the problem has been transformed into a large diversity instance of the original problem. One can easily see that any feasible solution to the integer program gives a feasible schedule for the problem by simply replacing tasks $k+1$ to $2k+2$ by idle times (or by nothing if all real tasks have been scheduled). Moreover, any feasible scheduled for $1|\text{FSE}|C_{\max}$ gives a feasible solution to the integer program because any idle time in the schedule can easily be decomposed as the sum of 1 or several idle times with durations 1 to $k+1$ since there are at most k FSE. However, the objective values of the two problems might differ. Hence, we need to show that any optimal solution to the IP gives an optimal schedule and vice-versa.

Remark that for any solution to the IP gives a schedule with makespan C^* and that any optimal schedule gives a solution to the IP with $C = C_{\max}$. Hence, $C^* \geq C_{\max}$ and $C^* \leq C_{\max}$. Therefore, $C^* = C_{\max}$.

Let π^* an optimal schedule. Convert it into a solution to the IP by setting m_{ij} values for real tasks according to π^*, decomposing idle times as a sum of feasible tasks with processing times in $1, \ldots, k+1$ and then set m_{ij} values for idle tasks. Set x_i and S_{jf} according to π^* and if $C(\pi^*) < \gamma_k$, finish with an additionnal task of kind $2k+2$. One can easily verify that this schedule is feasible.

If $C(\pi^*) > \gamma_k$, we have $C(\pi^*) = W_{k+1} = \max W_j$ and constraint (17) forces $y_j = 0 \ \forall j$. Hence we can simplify constraints (18) and (19) as the single constraint: $C \geq W_{k+1}$. Since C is not constrained in any other way, $C = W_{k+1} = C(\pi^*)$ is feasible.

Otherwise, let $j^* = \min\{j : C(\pi^*) < \gamma_j\}$ and set $y_j = 0$ for $j < j^*$ and $y_j = 1$ for $j \geq j^*$. This is feasible and $W_{j^*} = C(\pi^*)$. Moreover, since $C(\pi^*) \leq Q$ the right side of constraint (19) is ≤ 0 for $j > j^*$. Since W_j are non-decreasing, we can simplify constraints (18) and (19) as the single constraint: $C \geq W_{j^*}$. Since C is not constrained in any other way, $C = W_{j^*} = C(\pi^*)$ is feasible.

Therefore, $C^* \leq C(\pi^*)$.

Consider an optimal solution to the integer program. Because of the previous result and since $C_{\max} \leq Q$, $C^* \leq Q$. Remark that if $y_j = 0$ when $y_j = 1$ is feasible, setting $y_j = 1$ can only help improving the solution. Moreover, denote $j^* = \min\{j : y_j = 1 \text{ is feasible}\}$. Moreover, because of constraint (14), $y_j = 1$ is feasible for all $j \geq j^*$ and W_j are non-decreasing. Set $y_j = 1$ for $j \geq j^*$, this is feasible without modifying any other decision variable. Now, we can simplify constraints (18) and (19) as the single constraint: $C \geq W_{j^*}$ and since we the solution is minimized and C is not constrained in any other way, $C^* = W_{j^*}$. Now, let π be the schedule corresponding to current solution. π is feasible and $C(\pi) \leq W_{j^*} = C^*$, hence $C(\pi^*) \leq C^*$.

We have proved that solving this integer program gives an optimal schedule. Moreover, for fixed k, this program has a fixed number of variables and a fixed number of constraints. Therefore, we can apply Theorem 3, which proves Theorem 2. □

4.5 Complexity

Using Eisenbrand's algorithm to solve the integer program gives a linear time algorithm for fixed k.

References

[1] A.H. Abdekhodaee, A. Wirth, and H.S. Gan. Scheduling two parallel machines with a single server: the general case. *Computers & Operation Research*, 33:994–1009, 2006.

[2] J.-C. Billaut and F. Sourd. Single machine scheduling with forbidden start times. *4OR - A Quarterly Journal of Operations Research*, 7:37–50, 2009.

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

[4] N. Brauner, G. Finke, V. Lehoux-Lebacque, C. Rapine, H. Kellerer, C. Potts, and V. Strusevich. Operator non-availability periods. *4OR - A Quarterly Journal of Operations Research*, 7(3):239–253, 2008.

[5] T.C.E. Cheng, G. Wang, and C. Sriskandarajah. One-operator-two-machine flowshop scheduling with setup and dismounting times. *Computers & Operation Research*, 26:715–730, 1999.

[6] J.J. Clifford and M. E. Posner. Parallel machine scheduling with high multiplicity. *Mathematical Programming*, 89(3):359–383, 2001.

[7] C. Filippi and A. Agnetis. An asymptotically exact algorithm for the high-multiplicity bin packing problem. *Mathematical Programming*, 104:21–37, 2005.

[8] N.G. Hall, C. Potts, and C. Sriskandarajah. Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102:223–243, 2000.

[9] D.S. Hochbaum and R. Shamir. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Operations Research*, 39(4):648–653, 1991.

[10] C.P. Koulamas. Scheduling two parallel semiautomatic machines to minimize machine interference. *Computers & Operation Research*, 23(10):945–956, 1996.

[11] S.A. Kravchenko and F. Werner. Parallel machine scheduling problems with a single server. *Mathematical & Computer Modelling*, 26(12):1–11, 1997.

[12] V. Lebacque, N. Brauner, B. Celse, G. Finke, and C. Rapine. Planification d'expériences dans l'industrie chimique. In J.-F. Boujut, D. Llerena, and D. Brissaud, editors, *Les systèmes de production : applications interdisciplinaires et mutations*, pages 21–32. Hermès-Lavoisier, 2007. ISBN 978-2-7462-1819-2.

[13] C.-Y. Lee. Machine scheduling with availability constraints. In J.Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, pages 22–1 – 22–13. Chapman & Hall/CRC, London, 2004.

[14] J. Ou, X. Qi, and C.-Y. Lee. Parallel machine scheduling with multiple unloading servers. *Journal of Scheduling*, 13(3):213–226, 2009.

[15] C. Rapine, N. Brauner, G. Finke, V. Lebacque. Single Machine Scheduling with Small Operator-Non-Availability Periods, *Journal of Scheduling*, 15:127–139, 2012.

[16] C. Rapine, N. Brauner. Polynomial time algorithms for makespan minimization on one machine with forbidden start and completion times, *Les Cahiers Leibniz*, vol. 181, 2010.

[17] F. Eisenbrand. Fast integer programming in fixed dimension. *Lecture Notes in Computer Science, Algorithms - ESA*, pp196–207, 2003.