Just-in-time changeover cost minimization problem : Dynamic programming approach and complexity results

Lyès Benyoucef, Bernard Penz

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Just-in-time changeover cost minimization problem : Dynamic programming approach and complexity results

Lyès Benyoucef — Bernard Penz

N° 4727

Feb 2003

THÈME 4

ISSN 0249-6399 ISRN INRIA/RR--4727--FR+ENG

Just-in-time changeover cost minimization problem : Dynamic programming approach and complexity results

Lyès Benyoucef* , Bernard Penz †

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet MACSI

Rapport de recherche n° 4727 — Feb 2003 — 27 pages

Abstract: The problem studied in this paper originates from a just-in-time environment where a new supplying mode, called *Synchronous Delivery*, has been introduced recently. We consider a multi-product single machine scheduling problem in which the total changeover costs has to be minimized. The products are of different types and a changeover cost is incurred whenever switching from one product type to another. For each product, the processing time is equal to one and its deadline is different from deadlines of all other products. We present five complexity results and propose dynamic programming and heuristic approaches to the problem. Computational results are given comparing two heuristics to the optimal solution given by the dynamic programming method.

Key-words: Scheduling, single machine, changeover cost, complexity, dynamic programming approach, heuristic.

* INRIA-LORRAINE, MACSI Project. Email: lyes.benyoucef@loria.fr
† GILCO, ENSGI-INPG, 46 avenue Félix-Viallet, 38031 Grenoble cedex 1, France. Email: bernard.penz@gilco.inpg.fr

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Problème de minimisation du coût de changement de production en environnement de type Juste-à-Temps: Complexité et approche par programmation dynamique

Résumé : Le problème d'ordonnancement étudié dans cet article est observé dans un environnement de production et livraison de type juste-à-temps, où un nouveau mode d'approvisionnement, appelé *livraison synchrone*, a été récemment introduit. Nous considérons un problème d'ordonnancement sur une seule machine où on cherche à minimiser le coût de changement de production total. Les produits sont de différents types et un coût de changement est généré à chaque fois que la machine change de type de produit en production. Pour chaque produit, le temps de production est unitaire et un délai de livraison est à respecter. Principalement dû au contexte de la livraison synchrone, les délais de livraison sont différents. Liée à la nature du coût de changement, une étude de complexité de cinq versions du problème est présentée dans une première partie. Dans une seconde partie, une approche optimale par programmation dynamique est proposée. Des expériences numériques, comparant deux heuristiques à la solution optimale donnée par la méthode de programmation dynamique, sont présentées.

Mots-clés : Ordonnancement, machine unique, coût de changement, complexité, approche par programmation dynamique, heuristique.

1 Industrial context and problem description

More and more companies are focusing on optimization of their supply chains. This implies, among other things, an attempt to decrease stocks by focusing on the flow of goods. A new supplying mode, called *synchronous delivery*, was recently developed to achieve this objective. This mode was initially proposed in car industry to supply bulky components which are of great diversity, and which are very expensive to handle (see Benyoucef [1]).

In this mode, the components are delivered in the same order as they are used on assembly line. In car industry, an order for a component (car seats for example) is sent to a supplier, for example electronically, when a car enters assembly line. The supplier has to produce and then deliver the component following the order sequence, within a very short time limit. Production for a small local storage of size U is allowed, as it is known that all varieties of the component will eventually be demanded. The objective of the supplier is to schedule his production, respecting his own constraints (for example load balancing and a very limited storage), in order to minimize some criteria. These criteria often are the minimization of total production and storage costs, total changeover costs or the number of production changeovers.

These scheduling problems are dynamic since a new order may arrive each time a new car enters the assembly line. The supplier objective is now to develop good scheduling policies to manage his production (see Benyoucef et *al.* [3]). In the synchronous mode of delivery, assembly line (customer in our case) shares with supplier (of seats for example) some information concerning the sequence of deliveries. An approach to schedule the production is to calculate, at each decision time, the optimal solution of a static problem. Hence the name *iterative static*. The method calculates a schedule, minimizing the total changeover costs for the known future deliveries. Then we focus on the static problem.

In this paper, we assume that the whole sequence of deliveries (orders) of length T is communicated to the supplier (T orders denoted $\{J_1, \ldots, J_T\}$). The production system of the supplier, which can be viewed as a single machine, is able to produce N different product types, denoted $\{\mathcal{T}_1, \ldots, \mathcal{T}_N\}$, without preemption. Each order J_j corresponds to exactly one product unit, and empty orders, in the sense that any type of product is demanded, never occur. The order J_j contains the type of product ($J_j \in \{\mathcal{T}_1, \ldots, \mathcal{T}_N\}$) to be delivered exactly $\tilde{d}_j$ units later. The production time is equal to one unit and the deadline $\tilde{d}_j$ must be respected. Due to the synchronous delivery context, all the deadlines are different and the machine must not be idle in $[0, T]$. There is a changeover cost of switching from one product type $\mathcal{T}_i$ to another one $\mathcal{T}_j$ with $(i \neq j)$. The changeover cost may be product or sequence dependent. Even considering the changeover time is more realistic, for our case and in order to reduce the complexity of the general problem, we consider that the changeover time is negligible. The problem is to find a feasible schedule of orders that minimizes the total

changeover cost. Using the 3-field notation presented in (Blazewicz et *al.* [4]), our scheduling problem is denoted $1/p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\ \sum$ *Changeover costs.*

Remark 1 *For the following sections and without loss of generality, we replace the term order* J_j *by job* J_j *(for* $j = 1, \ldots, T$*).*

Related to the changeover and storage costs structure, we will consider the five following problems :

P1: $1/p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\sum$ *Sequence dependent changeover costs*;

P2: $1/p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\sum$ *Product dependent changeover costs*;

P3: $1/p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\sum$ *Unit changeover costs*;

P4: $1/r_j,\ p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\sum$ *Unit changeover costs*;

P5: $1/p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\sum$ *(Unit changeover costs + Product dependent storage costs)*;

Section 2 reviews literature on single machine scheduling problems with changeover cost minimization. In section 3, five complexity results are presented: 1) sequence dependent changeover cost, 2) product dependent changeover cost, 3) unit changeover cost, 4) unit changeover cost with product release time and 5) unit changeover cost and product dependent storage cost. In section 4, we give a description of the dynamic programming approach and two efficient heuristics where the changeover cost is product dependent. Computational results comparing the optimal solution and the two heuristics introduced in section 4 are presented in section 5.

2 Literature review

A great number of single machine scheduling problems are considered in the literature. Among of them, we focus on problems in which the total changeover cost has to be minimized.

The general problem, where the deadlines are arbitrary, was first introduced by Glassey-[13]. Glassey [13] considers a set of orders of different types to be scheduled on a single machine. For each order, one knows the type of product, the size of the batch requested and its deadline. This problem is more general than our synchronous delivery problem, since several products can share the same deadline. Each order must be delivered without any delay. The production time required for each unit of product is unit. The machine produces a unit of product in each unit of time. No preemption and idle time is allowed. Glassey proposes an implicit-enumeration method with dominance rules to solve the problem with

unit changeover costs.

Gascon and Leachman [15] propose a backward-time, implicit enumeration method similar to that of Glassey for solving a closely related problem including both changeover and holding costs. For product independent changeover cost and without holding costs, Mitsumori [15] proposes dynamic programming algorithms.

Driscol and Emmons [10] propose a dynamic programming algorithm for the Glassey problem with product dependent changeover costs. Hu, Kuo and Ruskey [14] present polynomial algorithms for Glassey problem with a particular changeover cost structure. They assume that a unit changeover cost is incurred if one changes production from type J_i to J_j with $i < j$ and no cost if $i > j$.

Blocher [5], Blocher and Chand [6] and Blocher et *al.* [7] consider different versions of the problem presented in Glassey [13]. Their studies are dedicated to both product dependent and sequence dependent changeover costs problems, with and without changeover times. They present an implicit enumeration method to solve the problem to optimality. Also, analytical results are proposed to reduce the number of solutions explored by this method.

Bruno and Downey [8] study the complexity of the changeover problem. They consider problems where a set-up time or a changeover cost are associated with switching from one product type to another, and different orders can have the same deadline. They propose a proof of *NP*-completeness in the ordinary sense for Glassey's problem. Also, they prove that various restricted versions of the general problem remain *NP*-complete. In the same paper, they consider particular cases of the problem where the changeover time is unit. For these cases, ordinary *NP*-complete are presented.

Schmidt [16] shows that when one considers a storage capacity dependent on the type of product, then the problem with three types of products is *NP*-hard, and he proposes an optimization polynomial algorithm in the case of two types of products.

In Benyoucef et *al.* [2], the authors present a linear time rule to solve the 2-product case when all the products have the same production time and different deadlines. To our knowledge the case where all deadlines are different has not been considered in the literature so far.

3 Complexity results

In this section, we present complexity results for the five scheduling problems denoted respectively by **P1**, **P2**, **P3**, **P4** and **P5**.

We will show that these problems are NP-hard. For this study, only problem **P1** is proved as NP-hard in the strong sense, problems **P2**, **P3**, **P4** and **P5** are NP-hard in the ordinary sense.

It is important to mention that starting from the NP-Hardness proof of problem **P3**, we can conclude directly that problems **P4** and **P5** are NP-hard. Indeed, the proof of problem **P3** is quite long and tedious, in this paper we present much simpler complexity proofs for problems **P4** and **P5**.

3.1 $1/p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\ \sum$ *Sequence dependent changeover costs*

Theorem 1 *The scheduling problem* $1/p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\ \sum$ *Sequence dependent changeover costs is NP-hard in the strong sense.*

Proof. The proof is obtained using the shortest Hamiltonian chain, which is NP-hard in the strong sense problem (see Garey and Johnson [11]).
Let consider an instance of the shortest Hamiltonian problem:

Instance: A complete undirected graph G with a distance matrix $\mathcal{M}$ (m_{ij} corresponds to the distance between nodes i and j).

Question: Does there exist a Hamiltonian chain with distance $\mathcal{D}$ or shorter?.

One builds an instance of the changeover problem as follows.

1. N product types $\{\mathcal{T}_1, \ldots, \mathcal{T}_N\}$.
2. N different single batch $\{J_1, J_2, \ldots, J_{N-1}, J_N\}$, with $J_j = \mathcal{T}_j$, $j = 1, \ldots, N$.
3. The changeover cost from the product type $\mathcal{T}_i$ to the product type $\mathcal{T}_j$ is equal to m_{ij}.

The deadlines of these delivery orders are represented in figure 1.

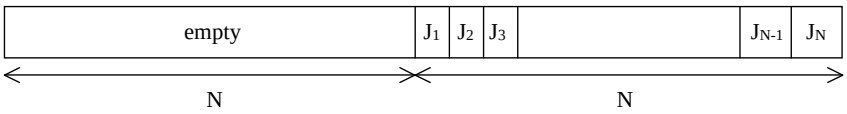

Figure 1: Instance of the problem with sequence dependent changeover cost

It is clear that we can schedule the batches $\{J_1, J_2, \ldots, J_{N-1}, J_N\}$ at a changeover cost $\mathcal{D}$ or less, if and only if there is a Hamiltonian chain not longer than $\mathcal{D}$. □

3.2 $1/p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\ \sum$ *Product dependent changeover costs*

Theorem 2 *The scheduling problem* $1/p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\ \sum$ *Product dependent changeover costs is NP-hard in the ordinary sense.*

Proof. This proof uses Partition problem (see Garey and Johnson [11]) as its point of departure. Let consider an instance of partition:

Instance: Finite set S of n positive values $\{w_1, w_2, \ldots, w_n\}$, such that $\sum_{i=1}^{n} w_i = 2K$.

Question: Does there exist a subset $\mathcal{A} \subset \mathcal{S}$ such that:

$$\sum_{\mathcal{A}} w_i = \sum_{\mathcal{S}-\mathcal{A}} w_j = K \quad ?$$

We present now the construction of an instance of the scheduling problem.

1. We have $N+1$ products types $\{\mathcal{T}_1, \ldots, \mathcal{T}_{N+1}\}$.
2. We have $2N+1$ batches to deliver.
3. We have to deliver a single batch J_1 (resp. J_2, J_3, ..., J_N) of product type $\mathcal{T}_1$ (resp. $\mathcal{T}_2$, ..., $\mathcal{T}_N$) by time $K+1$ (resp. $K+2$, $K+3$,..., $K+N$).
4. We have to deliver a batch J_{N+1} of product type $\mathcal{T}_{N+1}$ of size $K+N+1$ by time $2K+2N+1$.
5. We have to deliver a batch J_{N+2} (resp. J_{N+3}, J_{N+4}, ..., J_{2N+1}) of size w_1 (resp. w_2, w_3, ..., w_N) of product type $\mathcal{T}_1$ (resp. $\mathcal{T}_2$, ..., $\mathcal{T}_N$) by time $2K+2N+1+w_1$ (resp. $2K+2N+1+w_1+w_2$,..., $4K+2N+1$).
6. The changeover cost of the product type $\mathcal{T}_1$ (resp. $\mathcal{T}_2$, $\mathcal{T}_3$, ..., $\mathcal{T}_N$) is equal to w_1 (resp. w_2, w_3, ..., w_N). For the product type $\mathcal{T}_{N+1}$, the changeover cost is equal to 1.
7. The total changeover cost limit is $3K+1$.

This instance is represented on figure 2.
If the Partition problem admits a solution $\mathcal{A} = \alpha$ and $\mathcal{S} - \mathcal{A} = \beta$, then figure 3 shows how to obtain a solution with a total changeover cost equal to $3K+1$.

Now, consider a feasible schedule s, with changeover cost less or equal to $3K+1$. First, we observe that none of the first N jobs with deadlines $K+1$, $K+2$, ..., $K+N$ (see figure 2) can finish after the last, say job L, of $K+N+1$ jobs of type $\mathcal{T}_{N+1}$. Thus, all N types must be represented before L in s, and the changeover cost of s up to L is at least $2K+1$. Consequently, the changeover cost of s after L is at most K. This cost, however, can not

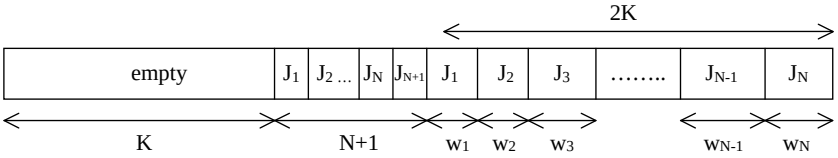

Figure 2: Instance of the problem with product dependent changeover cost

be less than K. Otherwise, more than K jobs would have to be moved before L making it late (notice that for each type $\mathcal{T}_1, \ldots, \mathcal{T}_N$, the number of jobs of the type equals the type's changeover cost plus one). Therefore, the cost must be exactly K, and consequently the set $\mathcal{A}$ including all types represented after L in s defines the required Partition. □

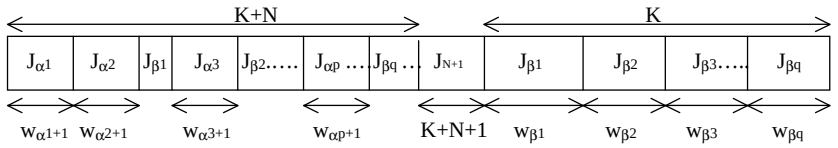

Figure 3: The schedule for the Partition $\alpha = \{\alpha_1, \ldots, \alpha_p\}$ and $\beta = \{\beta_1, \ldots, \beta_q\}$, where $q = N - p$

3.3 $1/p_j = 1$, $\tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i / \sum$ *Unit changeover costs*

The proof of problem **P3** is quite long and tedious. In Benyoucef [1], the proof is presented in more details.

This complexity proof uses a polynomial reduction starting from the logic POS-EXACT-3SAT problem known to be NP-Hard in the strong sense. This problem is a particular case of the traditional 3SAT problem, used in many complexity proves. In Benyoucef [1], we show how any instance of the POS-EXACT-3SAT problem can be transformed into a particular instance of the scheduling problem **P3**. The idea is that, if we know how to solve in polynomial time problem **P3**, then we will be able to solve the POS-EXACT-3SAT problem in polynomial time. The POS-EXACT-3SAT problem being NP-hard, that prove that problem **P3** is also NP-hard.

The proof, in Benyoucef [1], is divided in two parts. In the first part, we show that if the POS-EXACT-3SAT problem admits a feasible solution, then the instance of problem **P3** admits a feasible solution with a total number of changeovers equals to a certain value v. Using some technical lemmas, in the second part of the proof, the author shows that if

the instance of problem **P3** have a feasible solution with at most v changeovers, then this solution gives a feasible solution to the POS-EXACT-3SAT problem. He conclude that the scheduling problem **P3** is NP-hard. Basically because the sizes of the batches used in the proof are function of 2^n, the strong sense of the NP-Hardness is lose.

Theorem 3 *The scheduling problem* $1/p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\ \sum$ *Unit changeover costs is NP-hard in the ordinary sense.*

Proof. (see Benyoucef [1]). □

3.4 $1/r_j,\ p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\ \sum$ *Unit changeover costs*

We assume, for this scheduling problem, that for each order (job) J_j a release date r_j must be respected.

Theorem 4 *The scheduling problem* $1/r_j,\ p_j = 1,\ \tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/\ \sum$ *Unit changeover costs is NP-hard in the ordinary sense.*

Proof. This proof uses the 3-partition problem, which is known as NP-hard in the strong sense (see Garey and Johnson [11]), as its point of departure.

Let consider an instance of the 3-partition problem:

Instance: Finite sequence of $3K$ integer values denoted $\{\alpha_1, \alpha_2, \ldots, \alpha_{3K}\}$ and positive integer b, such that $\frac{b}{4} \leq \alpha_i \leq \frac{b}{2}$ for $i = 1, \ldots, 3K$ and $\sum_{i=1}^{3K} \alpha_i = Kb$.

Question: Does there exist K pairwise disjoint three-element subsets denoted $\mathcal{S}_l \subset \{1, \ldots, 3K\}$ such that:

$$\sum_{\mathcal{S}_l} \alpha_i = b, \quad l = 1, \ldots, K?$$

In this proof, the elements of the subset $\mathcal{S}_l$ are denoted respectively π_{3l-2}, π_{3l-1} and π_{3l} for $l = 1, \ldots, K$.

Now we present the construction of an instance of the scheduling problem.

1. We have $N = 3K + 1$ products types $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{3K}\}$ and A.
2. We have to deliver a total of $n = Kb + K(b+1)$ single batch $\{J_1, J_2, \ldots, J_{n-1}, J_n\}$.
3. We have to deliver α_l single batch of type $\mathcal{T}_l$ $(l = 1, \ldots, 3K)$.
4. We have to deliver $K(b+1)$ single batch of type A.
5. The total number of changeovers limit is $4K - 1$.

For this instance, the jobs deadlines are represented in figure 4. Only the product type A have a release date greater than 0. Indeed, for the first K single batch of type A the release dates r_j are equal to $\tilde{d}_j - 1$. The last Kb single batch of type A have a release date equal to $K(b+1)$. By assumption, all the product types $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{3K}\}$ are available at time 0. We can easily observe (see figure 4) that the total empty space corresponds to Kb single place (ie. K intervals of b single place, each place corresponds to a single batch).

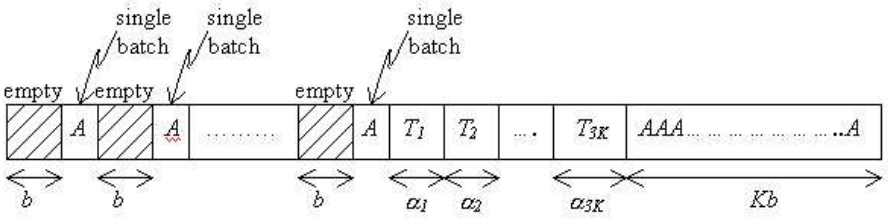

Figure 4: Instance of the scheduling problem with release dates

Does there exist a feasible schedule to our instance with a total number of changeovers less than $4K - 1$?

Let suppose first that the 3-partition problem admits a feasible solution. In this case, there exist K pairwise disjoint three-element subsets such as each subset contains b single batch of 3 different types. If we schedule each one of these K subsets in the K empty intervals of length b and we group the last Kb single batch of type A with the single batch of type A in the in K^{th} position (see figure 4), we obtain a feasible schedule with exactly $4K - 1$ changeovers (see figure 5).

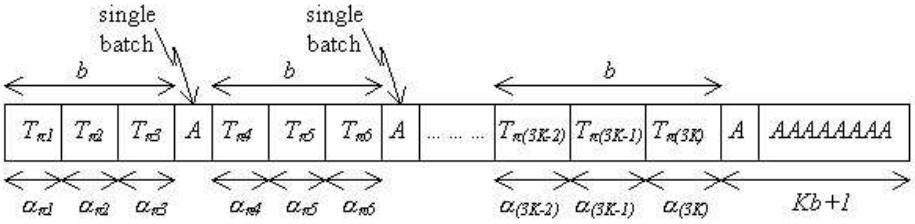

Figure 5: Feasible solution of the problem with $4K - 1$ changeovers

Now, let suppose that we have a feasible schedule to our instance with at most $4K - 1$ changeovers. We first note that for the product type A and related to the release dates, we have at least K batches of type A. In addition, there exists $3K$ different product types,

which imposes : 1- that the products of the same type are grouped in the same batch, and 2- that the last Kb single batch of type A are grouped with the K^{th} single batch of the same type. This last operation implies that batches of type $\mathcal{T}_j$ must be moved in the empty intervals, filling them exactly. Consequently, placing these batches in the K empty intervals, it comes to solve the 3-partition problem. Which conclude the proof. □

3.5 $1/p_j = 1$, $\tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/ \sum$ *(Unit changeover costs + Product dependent storage costs)*

For this scheduling problem, we consider product dependent storage costs in addition to the unit changeover cost.

Theorem 5 *The scheduling problem* $1/p_j = 1$, $\tilde{d}_j$ *integer, with* $\tilde{d}_j \neq \tilde{d}_i$ *for* $J_j \neq J_i/ \sum$ *Unit changeover costs + Product dependent storage costs is NP-hard in the ordinary sense.*

Proof. The proof is obtained using the 3-partition problem (presented above), which is NP-hard in the strong sense (see Garey and Johnson [11]). Without loss of generality, we assume that $b \geq K$.

We present now how an instance of our scheduling problem is built:

1. We have $N = 3K + 2$ products types $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{3K}\}$, A and B.
2. We have to deliver a total of $n = Kb + K + (Kb + 1)$ single batch $\{J_1, J_2, \ldots, J_{n-1}, J_n\}$.
3. We have to deliver α_l single batch of type $\mathcal{T}_l$ $(l = 1, \ldots, 3K)$.
4. We have to deliver K single batch of type A.
5. We have to deliver $Kb + 1$ single batch of type B.
6. The unit storage cost of product type A for one time unit is equal to 1. We do not consider storage costs for the rest of product types.
7. The total number of changeovers limit is $4K$.

The jobs deadlines are equal to the delivery times represented by figure 6.

Does there exist a feasible schedule to our instance with total (changeover + storage) costs less or equal to $4K$?

Initially, if there exist a feasible solution to the 3-partition problem, than it is easy to see (see figure 7) that we can built a feasible schedule with a total (changeover and storage) cost equals to $4K$.

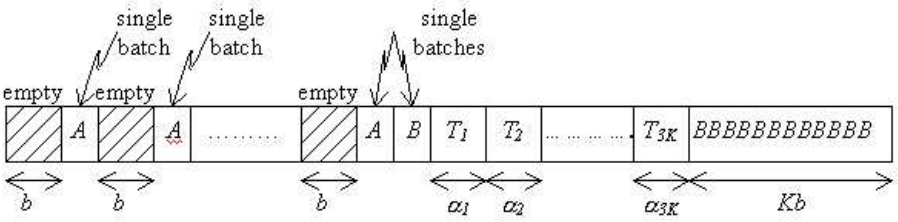

Figure 6: Instance of the problem with storage costs

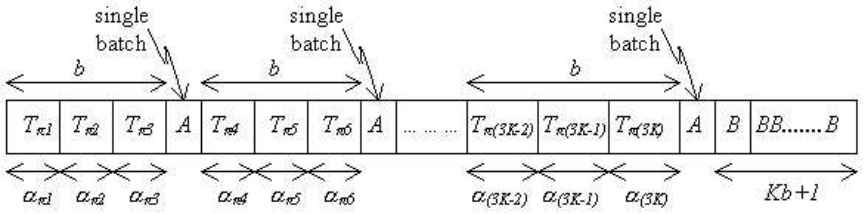

Figure 7: Feasible solution to the problem with storage costs

Let suppose now that we have a feasible solution to our scheduling problem with a total cost less or equal to $4J$. First of all, because we have $3K + 2$ different product (job) types, that inevitably involves a total cost at least equal to $3K + 1$. Due to the assumption that $b \geq K$, we can conclude that two single batch of type A cannot be grouped in the same batch without exceeding the required total cost $4K$. Consequently, related to the fact that we should have K single batch of type A in the required solution, the only possibility is that the products (jobs) of the same type are grouped in the same batch, and all the single batch of type A are scheduled just before their deadlines in order to not generate storage costs. Secondary, to built exactly one batch of type B, this involves that all the products type $\mathcal{T}_j$ must be moved and fill the free intervals. To move these batches in that intervals this require to have a feasible solution to the 3-partition problem. This conclude the proof. □

4 Dynamic programming approach

In this section we present a dynamic programming approach dedicate to the product dependent changeover cost problem (problem **P2**). This approach builds a state graph from time T to 0 (backward formulation). It works on product types rather then on jobs themselves. Optimal algorithm and two heuristics to solve the problem are presented.

In our modeling, for each product type we associate a chain of unit production time jobs (of the same type) with distinct deadlines. For example, consider the case with 3-product type denoted ($\mathcal{T}_1 = A$, $\mathcal{T}_2 B$ and $\mathcal{T}_3 = C$) and $T = 10$. We have to deliver 6 units of type A (resp. 2 of B and 2 of C) with deadlines $\{1, 4, 5, 9, 11, 12\}$ (resp. $\{8, 10\}$ and $\{6, 7\}$). Thus, we can associate chain precedence constraints between the jobs of the same type without changing the feasibility of the solutions. For instance, if A with deadline 1 is scheduled after A with deadline 4, then their exchange is always feasible.

Hence, products (jobs) of the same type are linked in chain and a job precedes all the jobs which have greater deadlines. We present an instance of the problem in figure 8. The figure depicts a problem with 4-product type denoted $\mathcal{T}_1 = A$, $\mathcal{T}_2 B$, $\mathcal{T}_3 = C$ and $\mathcal{T}_4 = D$, $T = 13$, and the delivery sequence $(J_1, J_2, \ldots, J_{13})$ is equal to $(A\ C\ D\ A\ B\ D\ C\ D\ A\ B\ B\ B\ D)$ with deadlines respectively $(2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)$. Each job (order) J_i corresponds to a single batch in $\{A, B, C, D\}$. Using this representation, deadlines can be modified as follows: the last job J_{13} must be finished at time 13 (we are in a unit production time situation and exactly 13 jobs ($T = 13$) have to be produced without interruption). For example, the modified deadline of job J_{12} (single batch of type B) with deadline 15 is then 13. Considering the precedence constraints, the previous single batch B has a modified deadline 12 and the single batch before a modified deadline 11. Consequently, the sequence to produce and to deliver $(A\ C\ D\ A\ B\ D\ C\ D\ A\ B\ B\ B\ D)$ should respect the deadlines sequence given by $(2, 5, 6, 7, 8, 9, 10, 11, 12, 11, 12, 13, 13)$. This transformation is depicted in figure 8.

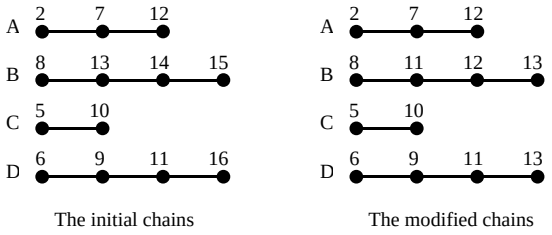

Figure 8: Chains representation of a sequence to deliver

Notations

Let us define the following notations:

1. $x_i(t)$: cumulative production of product type $\mathcal{T}_i$ in $[t, T]$, $i = 1, \ldots, N$,
2. $x(t) = (x_1(t), x_2(t), \ldots, x_N(t))$: vector of cumulative production at time t (called a state or a node of the graph),

3. $n(x(t))$: total changeover costs incurred in $[t, T]$ for the state $x(t)$,
4. $r_i(x(t))$: longest batch of jobs of type $\mathcal{T}_i$ that can complete without deadline violation at t from a given state $x(t)$,
5. $|r_i(x(t))|$: size of batch $r_i(x(t))$,
6. n_i: number of unit jobs of type $\mathcal{T}_i$ to schedule in the sequence,
7. c_i: changeover cost of the product type $\mathcal{T}_i$.

4.1 The dominance rules

We found that the following three rules substantially reduce the number of nodes in the state graph.

Rule 1 *Always schedule a* complete *batch $r_i(x(t))$, for some product type $\mathcal{T}_i$, next.*

Proof. Let consider $x(t)$ a state of the graph and $\mathcal{T}_i$ a product type with $r_i(x(t))'0$. We denote by $x(t_1) = x(t - r_i(x(t)))$ (resp. $x(t_2) = x(t - r_i(x(t) + k))$ the state obtained if we decide to schedule all the batch $r_i(x(t))$ (resp. a part of the batch $r_i(x(t))$ denoted by $r_i(x(t)) - k$ with $k`r_i(x(t))$). Consequently, we have $t_1 < t_2$ with $x(t_1) > x(t_2)$ (ie. $x_j(t_1) = x_j(t_2)$ for each $j \neq i$ and $x_i(t_1) > x_i(t_2)$) and $n(x(t_1)) = n(x(t_2))$. We need to prove that state $x(t_1))$ dominates state $x(t_2))$.

Initially, we can make the following observation. If the optimal path from state $x(0)$ to state $x(T)$ (optimal scheduling) contains state $x(t_2)$ then we can take out, from this path between 0 and t_2, a batch of type $\mathcal{T}_i$ of size k (ie. we create a empty space of length k between 0 and t_2, which correspond to k single batch). After that, we move all the batches between 0 and t_2 of k units starting from t_2. After that, between t_2 and t_1 we replace the empty space by k single batch of type $\mathcal{T}_i$ to obtain state $x(t_1)$. Hence, we built a new path (a new schedule) which have, in the worst case, a total changeover costs equal to that given by the initial path (path which contains state $x(t_2)$ and optimal by assumption). From where the proof that $x(t_1)$ dominates $x(t_2)$. □

Example 1 *We have to produce and to deliver a sequence $(ABABA)$, with deadlines respectively $(2, 3, 4, 5, 6)$. Using the chains representation procedure with $T = 5$, deadlines can be modified to have $(2, 3, 4, 5, 5)$. In figure 9, we present the evolution of the states graph when Rule 1 is applied.*

Rule 2 *Always schedule batch $r_i(x(t))$ that* exhausts *some product type $\mathcal{T}_i$, i.e. such that $|r_i(x(t))| + x_i(t) = n_i$, first.*

Proof. Let suppose that we have a state $x(t)$ and a product type $\mathcal{T}_i$ with $|r_i(x(t))| + x_i(t) = n_i$. We denote by $x(t_1)$ the state obtained when we decide to schedule the total batch

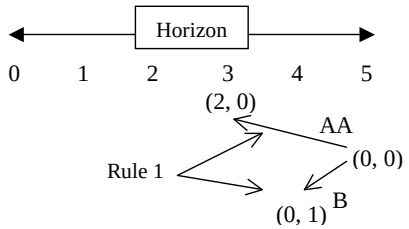

Figure 9: Example of Rule 1 application

$r_i(x(t))$ of type $\mathcal{T}_i$, with $t_1 = t - r_i(x(t))$. Also, we suppose that we have a product type $\mathcal{T}_j$ with $|r_j(x(t))| + x_j(t) < n_j$. >From the state $x(t)$, we decide to schedule another batch of size $r_j(x(t)$ of type $\mathcal{T}_j$ to obtain the state $x(t_2)$, with $t_2 = t - r_j(x(t)$. We need to prove that state $x(t_1)$ dominates state $x(t_2)$.

If we consider that the optimal path, from $x(T)$ to $x(0)$, contains state $x(t_2)$, then we can built a path which contains state $x(t_1)$ with at most the same total changeover costs as the *optimal* path. In fact, from the optimal path (which contains state $x(t_2)$ by assumption), it is possible to remove (between 0 and t_2) a batch of size $r_i(x(t))$ (to have $r_i(x(t))$ empty single spaces), then we push the batches between 0 and t starting from t to 0. After that, we replace the empty space by the batch $r_i(x(t))$, to obtain the state $x(t_1)$ and the path between $x(t_1)$ and $x(0)$. By this operation, the new path (which contains state $x(t_1)$) has a total changeover costs lower or equal to that which contains $x(t_2)$ (optimal by assumption). Consequently, we built a new path (schedule) with a total changeover costs equal or less than the optimal one, which prove that state $x(t_1)$ dominates state $x(t_2)$. □

Example 2 *We have to schedule the same sequence presented in example 1. Figure 10 depicts the evolution of the states graph when Rule 2 is applied.*

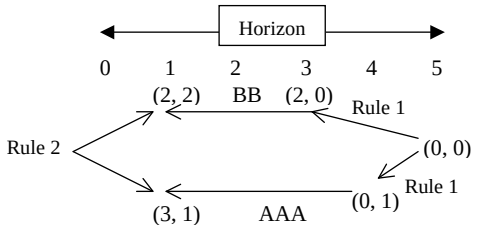

Figure 10: Example of Rule 2 application

If two or more types satisfy this property, we schedule any of them first.

Rule 3 *Let consider two states $x(t_1)$ and $x(t_2)$ withs the following assumptions:*

- $t_1 \leq t_2$
- $x(t_1) \geq x(t_2)$ *i.e.* $x_i(t_1) \geq x_i(t_2)$ *for all* $\mathcal{T}_i$
- $n(x(t_1)) \leq n(x(t_2))$

then the state $x(t_1)$ *dominates the state* $x(t_2)$.

Proof. The proof is given by contradiction. Let consider that the total changeover costs between states $x(t_1)$ and $x(0)$ is equal to a and the total changeover costs between states $x(t_2)$ and $x(0)$ equal to b. We assume that $n(x(t_1)) \leq n(x(t_2))$.

Now, by assumption we consider that the path (schedule), which contains state $x(t_2)$, is strictly better than that which contains $x(t_1)$. Than $n(x(t_1)) + a > n(x(t_2)) + b$, and $a > b$. Let take an optimal sequence of batches between $x(0)$ and $x(t_2)$, which realise a total changeover costs equal to b. For each product type $\mathcal{T}_i$, let us remove from this sequence $x_i(t_1) - x_i(t_2)$ products having the height starting time (starting production time). This sequence corresponds to a feasible schedule for all products which are scheduled between states $x(t_1)$ and $x(0)$, and its total changeover costs is lower or equal to b. Hence, we built (between states $x(t_1)$ and $x(0)$) a new feasible batch sequence with a total changeover costs $b' \leq b$. This represents a contradiction with the fact that $a > b$. Consequently, state $x(t_1)$ dominates state $x(t_2)$. □

Example 3 *We have to deliver a sequence* $(BAABA)$ *with deadlines respectively (1,2,4,5,6). Deadlines can be modified as in example 1 to have (1,2,4,5,5). The changeover cost for the product type A (resp. B) is* 1 *(resp.* 2*). Figure 11 presents the evolution of the states graph when Rule 3 is applied.*

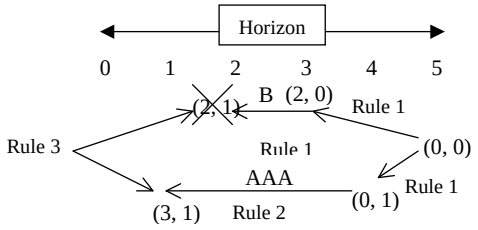

Figure 11: Example of Rule 3 application

4.2 Optimal algorithm and heuristics

In figure 12, we give the description of the dynamic programming algorithm, of complexity $O(\prod n_i)$, that leads to an optimal solution.

In figures 13, 14, to solve the problem we present two heuristics denoted $H1$ and $H2$ of complexity $O(NT)$. The heuristic $H1$ uses only Rules 1 and 2, when building a path from T to 0. The principle of the heuristic $H2$ is the same that $H1$, but applied in the forward sense. With the forward formulation, we have the following notations:

- $y_i(t)$: cumulative production of the product type $\mathcal{T}_i$ in [0, t], $i = 1, \ldots, N$
- $y(t) = (y_1(t), y_2(t), \ldots, y_N(t))$: vector of cumulative production at time t, $t = 0, \ldots, T$.

Remark 2 *We denote by S (resp. S' and S'') the states space obtained using the optimal algorithm (resp. heuristics $H1$ and $H2$).*

Algorithm

Step 0. Initialise $x(T) = 0$, $n(x(T)) = 0$ and put $x(T)$ in the state space S

Step 1. While S does not contain $x(0)$, for each state $x(t) \in S$ with maximum t and which has not been branched out do

1. Calculate $r_i(x(t))$ and $|r_i(x(t))|$ for all i
2. If there is a product type with $r_i^*(x(t))$ as in Rule 2, then
   - Schedule the batch $r_i^*(x(t))$
   - If state $x(t-|r_i^*(x(t))|) \in S$, then compute $n(x(t-|r_i^*(x(t))|)) = min\{n(x(t-|r_i^*(x(t))|)), n(x(t)) + c_i^*\}$
   - else add state $x(t-|r_i^*(x(t))|)$ to S, and compute $n(x(t-|r_i^*(x(t))|)) = n(x(t)) + c_i^*$
3. else
   - Branch out $x(t)$ using all non-empty $r_i(x(t))$'s, Rule 1
   - For each new state $x(t-|r_i(x(t))|)$ do
     (a) If state $x(t-|r_i(x(t))|) \in S$, then compute $n(x(t-|r_i(x(t))|)) = min\{n(x(t-|r_i(x(t))|)), n(x(t)) + c_i\}$
     (b) else add state $x(t-|r_i(x(t))|)$ to S, and compute $n(x(t-|r_i(x(t))|)) = n(x(t)) + c_i$
4. Apply Rule 3 to S, if possible
5. Go to Step 1

Step 2. Build the schedule moving along the shorter path from $x(0)$ to $x(T)$

Figure 12: The optimal dynamic programming algorithm O

Heuristic H1

Step 0. Initialise $x(T) = 0$, $n(x(T)) = 0$ and put $x(T)$ in the state space S'

Step 1. While S' does not contain $x(0)$ do for the last state $x(t) \in S'$

(1) Calculate $r_i(x(t))$ and $|r_i(x(t))|$ for all i

(2) If there is $r_i^*(x(t))$ as in Rule 2, then

- Schedule $r_i^*(x(t))$ and add state $x(t-|r_i^*(x(t))|)$ to S'
- Compute $n(x(t - |r_i^*(x(t))|) = n(x(t)) + c_i^*$

(3) else

- Schedule the batch with maximum $|r_i(x(t))|$, using Rule 1, and add state $x(t - |r_i(x(t))|)$ to S'
- Compute $n(x(t - |r_i(x(t))|) = n(x(t)) + c_i$

(4) Go to Step 1

Step 2. Build the schedule moving from $x(0)$ to $x(T)$

Figure 13: The backward heuristic H1

Heuristic H2

Step 0. Initialise $y(0) = 0$, $n(y(0)) = 0$ and put $y(0)$ in the state space S''

Step 1. While S'' does not contain $y(T)$ do for the last state $y(t) \in S''$

(1) Calculate $r_i(y(t))$ and $|r_i(y(t))|$ for all i

(2) If there is $r_i^*(y(t))$ as in Rule 2, then

- Schedule $r_i^*(y(t))$ and add state $y(t + |r_i^*(y(t))|)$ to S''
- Compute $n(y(t + |r_i^*(y(t))|) = n(y(t)) + c_i^*$

(3) else

- Schedule the batch with maximum $|r_i(y(t))|$, using Rule 1, and add state $y(t + |r_i(y(t))|)$ to S''
- Compute $n(y(t + |r_i(y(t))|) = n(y(t)) + c_i$

(4) Go to Step 1

Step 2. Build the schedule moving from $y(0)$ to $y(T)$

Figure 14: The forward heuristic H2

If we applied heuristics $H1$ and $H2$ to the instance of the problem presented in figure 8, the schedules obtained are represented in figures 15 and 16 respectively.

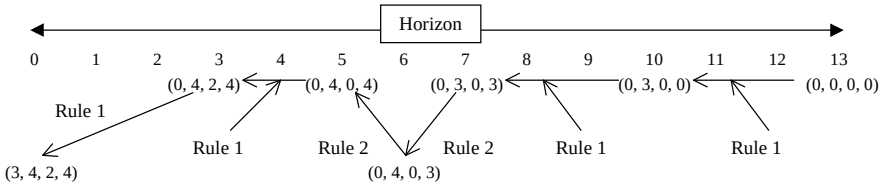

Figure 15: Graph generated by heuristic H1

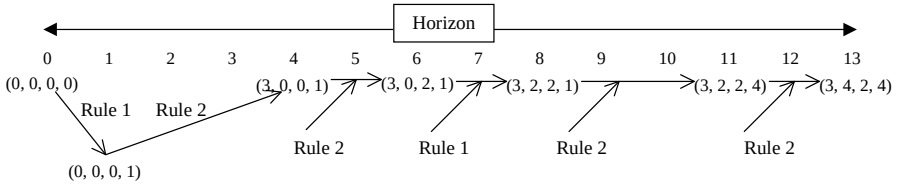

Figure 16: Graph generated by heuristic H2

5 Computational experiments

In this section computational experiments are presented where we compare results given by the optimization dynamic programming algorithm O and the heuristics $H1$ and $H2$. Two cases related to the changeover cost structure are considered. Indeed, we focus initially on the case where we have a unit changeover cost. In this case, minimizing the total changeover costs is equivalent to minimizing the number of changeovers. After that, we consider the case where the changeover cost is product dependent.

The objective is the minimization of the total changeover costs incurred during the day for example, which represent our horizon length T. Daily production and delivery can be formulated as an instance of our scheduling problem. Let us assume that the production system must deliver T units of N different product types daily, where the processing time is unitary for each one. Without initial stock the sequence to produce is given by the sequence to deliver. Generally, the production starts with a certain initial stock level of each product. The storage capacity U is less than T.

The instances we generate correspond to a daily production management. Each day the production starts with a stock of finished products made the day before. These products are used generally to furnish the first orders of the previous day. The production planned for the day permits to deliver the products ordered and not issued from the stock. A part of this production (U products) is made to fill up the stock for the following day.

Let consider an instance of the problem where $T = 6$, $N = 3$ ($\mathcal{T}_1 = A$, $\mathcal{T}_2 = B$ and $\mathcal{T}_3 = C$), the sequence to deliver is $A\ A\ B\ A\ C\ B$ with initial stock structure 1 unit of A and 1 unit of B ($U = 2$), we want to have at the end of the day, for example, 2 units of C. First we eliminate from the sequence A and B (the two products in the stock). We have then $0\ A\ 0\ A\ C\ B$ ($A\ A\ C\ B$ with deadlines 2 4 5 6). Afterwards, in order to have the 2 units of C in the stock at the end of the day, we introduce these products in the delivery sequence $A\ A\ C\ B\ C\ C$ with deadlines 2 4 5 6 6 6. We modified the deadlines according to the procedure explained before.

For these computational experiments, the sequences are generated using the discrete distribution with probability $1/N$ for each product type. Also, the U units to produce are given by the same probability distribution.

5.1 Unit changeover cost

The computational experiments presented in this section are dedicated to the unit changeover cost situation. The average number of changeovers during the day for different values of T (15, 30, 60), N (3, 4, 6, 8) and U (3, 5, 10), are given in table 2 (see appendix). In these table, the optimal solution is denoted by O and $Min(H1, H2)$ represents the minimum between $H1$ and $H2$. The average relative errors between O and $H1$, O and $H2$ and O and the $Min(H1, H2)$ are denoted respectively E_1, E_2 and E_3. They are computed as flow:

$$E_1(\%) = \frac{1}{Number\ of\ instances} \sum \frac{H1 - O}{O}$$

$$E_2(\%) = \frac{1}{Number\ of\ instances} \sum \frac{H2 - O}{O}$$

$$E_3(\%) = \frac{1}{Number\ of\ instances} \sum \frac{min(H1, H2) - O}{O}$$

In these experiments, the number of generated instances is between 1000 and 10000. Some remarks can be made:

- For all the experiments, we note that heuristic $H1$ is better than heuristic $H2$.
- Even if the difference between $H1$ and $H2$ is not significant for T=15, it becomes quite significant when the number of product types N increases. Again, when T increases (30, 60,...) in average $H2$ becomes worse than $H1$.
- The difference between O and $Min(H1, H2)$ is quite small. On average, the relative error is between 1.3% and 14.3%,which proves that the heuristics are quite accurate.
- We observe, in all the experiments, that the difference between the average errors given by $H1$ (E_1) and by $Min(H1, H2)$ (E_3), even for small values of T (15) are between 2% and 6%. These differences become negligible when T increases (30, 60...).

In conclusion, we showed that it is interesting to use heuristic $Min(H1, H2)$ when the sequence length T is shorter (<30), but in the cases with large values of T ($\geq$ 30) the difference between $Min(H1, H2)$ and $H1$ are not significant. For this last reason, it is better to use heuristic $H1$ when $T \geq 30$.

5.2 Product dependent changeover costs

In this section computational experiments are presented to compare results given by the optimization dynamic programming algorithm O and the heuristic $H1$, where the changeover cost is product dependent. In fact, the changeover cost, for each product type, is given by a continuous uniform distribution in the interval $[1, 10]$.

For different values of T (15, 30, 60), N (4, 6, 8) and U (3, 5, 10), the average relative error between the optimal cost O and the cost given by the heuristic $H1$, and the average number of nodes (states) generated by the optimal algorithm O are presented in table 1. In this table, the average relative error $E_1(\%)$ and the average number of nodes $AvNode$ are computed, where :

$$AvNode = \frac{1}{Number\ of\ instances} \sum Number\ of\ Nodes$$

The number of generated instances is 10000 for $T = 15$, 5000 for $T = 30$ and 1000 for $T = 60$.

For these computational experiments, some remarks can be made:

1. We observe that on average the difference between H and O is quite significant. Indeed, $E_1(\%)$ is less than 12% for $T = 15$ and increases to 13% and 25% when T increase to 30 and 60 respectively. For product independent changeover costs (see appendix, table 2), we observe that $E_1(\%) = 3.1\%$ for $T = 15$ and $U = 10$ and $E_1(\%) = 19.5\%$ for $T = 60$ and $U = 3$. Our computational experiments show $E_1(\%) = 4.9\%$ for $T = 15$ and $U = 10$ and $E_1(\%) = 25\%$ for $T = 60$ and $U = 3$. Thus heuristic $H1$ does not seem to be sensitive to changeover costs.
2. We observe that when the size of U increase the relative error $E_1(\%)$ decreases.
3. We observe that the $AvNode$ seems to be a concave function of U with its maximum reached for the value U/T equal to $1/5$ for $T = 15$, and $1/6$ for $T = 30$, and 60.

T=15									
	N=4			N=6			N=8		
	U=3	U=5	U=10	U=3	U=5	U=10	U=3	U=5	U=10
AvRE(%)	12	11.3	5.5	10.87	9.5	5.1	9.3	8.2	4.9
AvNode	27	19	7	37	29	12	37	31	17
T=30									
	N=4			N=6			N=8		
	U=3	U=5	U=10	U=3	U=5	U=10	U=3	U=5	U=10
AvRE(%)	16.5	16.4	15.5	18.2	17.6	14.3	18.4	16.3	13.1
AvNode	129	138	50	286	434	187	359	616	349
T=60									
	N=4			N=6			N=8		
	U=3	U=5	U=10	U=3	U=5	U=10	U=3	U=5	U=10
AvRE(%)	18.8	19.46	14.0	22.8	23	21	25	24	20.8
AvNode	400	687	926	1205	3478	9970	2243	8367	33200

Table 1: Average relative errors between optimal and heuristic costs, and average number of nodes generated by the optimal algorithm

6 Conclusions and further research

In this paper, we consider the multi-product single machine scheduling problem in which the total changeover cost has to be minimized. The jobs (products) are of different types. For each product, the processing time is equal to one and its deadline is different from deadlines of all other jobs. Our objective is to develop an efficient approach to schedule the jobs without deadlines violation and to minimize the total changeover costs.

The first part of this paper is basically dedicate to the complexity study. Related to the changeover cost structure, five complexity results are presented: 1) sequence dependent changeover cost, 2) product dependent changeover cost, 3) unit changeover cost, 4) unit changeover cost with product release time and 5) unit changeover cost and product dependent storage cost.

The second part of this paper is devoted to the resolution methods development. We present an optimal dynamic programming approach for *Product-Dependent* changeover cost minimisation. Two efficient heuristics, to approach the optimal solution, are presented. By simulation, these two heuristics are tested and comparing to the optimal solution where daily production and delivery context is considered. Our computational results proved that

the proposed heuristics are quite efficient and accurate.

Four natural perspectives for this work are: 1- simplify the complexity proof of problem **P3** and verify the strong sense of its NP-hardness proof, 2- comparison, for product dependent changeover cost, between H and some other heuristics presented in the literature (e.g. Drexl and Kimms [9]), 3- development and improvement of more general dominance rules and 4- compare the optimal algorithm and the two heuristics on large sequences (T and U very large).

References

[1] L. Benyoucef (2000). Résolution d'un problème d'ordonnancement dynamique d'un fournisseur dans un mode d'approvisionnement de type "Livraison synchrone". Ph.D. Dissertation, INPG of Grenoble (France).

[2] L. Benyoucef, Y. Frein and B. Penz (2001). Optimal solution for a two-product dynamic scheduling problem in a just-in-time environment. Int. J. Production Economics 74, 85-91.

[3] L. Benyoucef, Y. Frein, B. Penz, S.W. Wallace (2003). Synchronous Delivery : New Dynamic Scheduling Problems. To appear in "Journal of Scheduling" (JOS).

[4] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz (1996), Scheduling Computers and Manufacturing Processes, *Springer*.

[5] J.D. Blocher (1992). Single machine scheduling changeover problem. Ph.D. Dissertation, Purdue University (USA).

[6] J.D. Blocher and S. Chand (1996). A forward branch-and-search algorithm and forecast horizon results for the changeover-scheduling problem. European J. Oper. Res. 91, 456-470.

[7] J.D. Blocher, S. Chand and K. Sengupta (1999). The changeover scheduling problem with time and cost considerations: Analytical results and a forward algorithm. Oper. Res. 47, 559-569.

[8] J. Bruno and P. Downey (1978). Complexity of task sequencing with deadlines, set-up times and changeover costs. SIAM Journal on Computing 7, 393-404.

[9] A. Drexl and A. Kimms (1997). Lot sizing and scheduling : Survey and extensions. European J. Oper. Res. 99, 221-235.

[10] W.C. Driscoll and H. Emmons (1977). Scheduling Production on One Machine with Changeover Costs. AIIE Trans. 9, 388-395.

[11] M. Garey and D. Johnson (1979). Computers and Intractability: A guide to the theory of NP-completeness. Freeman and Company, New York.

[12] A. Gascon and R.C. Leachman (1988). A dynamic programming solution to the dynamic multi-item, single machine scheduling problem. Oper. Res. 36, 50-56.

[13] C.R. Glassey (1968). Minimum Changeover Scheduling of Several Products on one Machine. Oper. Res. 16, 342-352.

[14] T.C. Hu, Y.S. Kuo and F. Ruskey (1987). Some optimum algorithms for scheduling problems with changeover costs. Oper. Res. 35, 94-99.

[15] S. Mitsumori (1972). Optimum production scheduling of multi-commodity in flow line. IEEE Transactions 2, 486-493.

[16] G. Schmidt (1992). Minimizing changeover costs on a single machine in: W. Buhler, F. Feichtinger, F-J Radermacher, P. Feichtinger (eds.). DGOR Proceedings 90, Vol.1, Springer, 425-432.

Table 2: Comparison between optimal and heuristics average number of changeovers and relative errors

T=15												
	N=3			N=4			N=6			N=8		
	U=3	U=5	U=10	U=3	U=5	U=10	U=3	U=5	U=10	U=3	U=5	U=10
O	4.066	3.149	2.277	5.483	4.385	3.386	7.620	6.449	5.257	9.110	7.949	6.714
H_1	4.426	3.424	2.379	5.950	4.749	3.542	8.181	6.873	5.461	9.718	8.396	6.939
H_2	4.573	3.483	2.468	6.343	5.050	3.863	8.606	7.374	5.883	10.021	8.875	7.401
$Min(H_1, H_2)$	4.182	3.224	2.285	5.714	4.545	3.425	7.919	6.671	5.333	9.408	8.176	6.807
$E_1(\%)$	8.7	8.4	3.3	8.4	8.2	3.9	7.3	6.3	3.5	6.7	5.4	3.1
$E_2(\%)$	12.9	11.1	9.1	16.4	15.7	15.2	13.7	15.1	12.5	10.8	12.3	10.7
$E_3(\%)$	2.8	2.3	0.2	4.1	3.5	0.9	3.9	3.4	1.3	3.3	2.8	1.3
T=30												
	N=3			N=4			N=6			N=8		
	U=3	U=5	U=10	U=3	U=5	U=10	U=3	U=5	U=10	U=3	U=5	U=10
O	7.129	5.252	3.922	9.444	7.247	4.835	13.070	10.382	7.338	15.701	12.924	9.640
H_1	7.813	5.760	4.316	10.510	8.008	5.259	14.770	11.602	7.952	17.784	14.399	10.428
H_2	8.593	6.302	4.629	11.889	9.208	5.924	16.069	13.051	9.055	18.751	15.842	11.726
$Min(H_1, H_2)$	7.555	5.530	4.103	10.284	7.818	5.106	14.422	11.382	7.817	17.307	14.133	10.276
$E_1(\%)$	9.7	9.8	10.1	11.3	10.6	8.7	13	11.7	8.2	13.3	11.3	8
$E_2(\%)$	21.1	20.4	18.2	26.5	27.6	23	23.5	26.2	23.8	20	23.1	22
$E_3(\%)$	6	5.3	4.5	9	8	5.6	10.4	9.6	6.4	10.3	9.3	6.5
T=60												
	N=3			N=4			N=6			N=8		
	U=3	U=5	U=10	U=3	U=5	U=10	U=3	U=5	U=10	U=3	U=5	U=10
O	13.219	9.500	5.870	17.499	12.937	8.240	23.820	18.340	12.166	28.473	22.687	15.760
H_1	14.596	10.440	6.408	19.893	14.546	9.112	27.864	21.244	13.696	34.026	26 .518	18.100
H_2	16.589	12.010	7.314	23.384	17.524	11.197	31.628	25.410	17.057	36.914	3 0.915	21.910
$Min(H_1, H_2)$	14.384	10.266	6.235	19.760	14.454	9.040	27.639	21.125	13.642	33.558	26.400	18.010
$E_1(\%)$	10.5	10	9.3	13.7	12.5	10.6	17	15.9	12.6	19.5	16.9	14.8
$E_2(\%)$	25.8	26.8	24.9	34	35.8	36.2	33.1	38.9	40.5	30	36.6	39.3
$E_3(\%)$	8.8	8.1	6.3	13	11.8	9.8	16.1	15.4	12.2	17.9	16.4	14.3

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399