An Algorithm to deal with due date violation in a Multi-objective Scheduling problem

Francisco Ibáñez, Daniel Díaz Araya, Germán Zavalla, Raymundo Forradellas
LISI – Laboratorio Integrado de Sistemas Inteligentes
IdeI - Instituto de Informática - Universidad Nacional de San Juan
Cereseto y Meglioli – 5400 San Juan – Argentina
Tel: +54 264 426 47 21 - Fax: +54 264 426 51 01
{fibanez, ddiaz, kike}@iinfo.unsj.edu.ar

Abstract

This paper includes part of the strategies used to solve a scheduling problem developed for a company that produces flexible packaging, presented in quite a general form though. In this problem it is necessary to schedule several jobs that involve four process and for each one of them there is a group of machines available (of similar characteristics). Each activity is performed on just one machine. Besides, for our application, the scheduling must try to verify certain conditions. For each process (and consequently for all the activities that performs this process) there is a list of attributes.

The problem is not only to assign each activity to a starting time and to a specific machine, but also to try to verify conditions that depend on the values of the attributes of the activities. Moreover, there are criteria to choose a particular machine.

An approach to solve this problem was presented first in (Ibañez *et al*., 2001). As mentioned there, some jobs could not be fulfilled to meet their due dates. An approach to decrease the quantity of due dates violations was presented in (Ibañez *et al*., 2002). The algorithm presented in (Ibañez *et al*., 2001) is entirely dedicated to verify as many conditions as possible disregarding due date violations. The algorithm shown in (Ibañez *et al*., 2002) was focussed to reduce the number of due date violations by paying the price of decreasing the fulfilment of conditions. Roughly speaking, the first approach favours the company whereas the second one is more convenient for the customers.

The present work includes an algorithm, which allows us to assign weights to set an appropriate trade of between due date violation reduction and fulfillment of conditions.

Key words: Scheduling, Multi-objective Combinatorial Problems, Constraints Satisfaction

1. Introduction

This paper includes part of the strategies used to solve a scheduling problem developed for a company that produces flexible packaging. The application has been implemented in C++, employing routines of Ilog (Ilog, 1999). In this problem it is necessary to schedule several jobs. These jobs involve four process: Printing, Laminating, Cutting and Packing and for each one of them there is a group of machines available (of similar characteristics).

Each job is described by a list of four activities of given processing times, that perform the mentioned processes in that order. Each activity is performed on just one machine. For example, if *a* represents a printing activity and $\{M_1, ...,M_k\}$ represent the set of machines capable of executing the printing process, *a* will be performed by a member of the set $\{M_1, ...,M_k\}$. For our application the scheduling must also try to verify certain conditions.

For each process (and for all the activities that perform this process) there is a list of attributes. For the printing process, the attributes are: *ink line, duration of the (printing) process*, etc. These attributes are also associated to the machines but their values depend on the time. For each printing machine $M_1, ...,M_k$, the values of the attributes at time *t* are defined as equal to the values of the attributes of the activity that is being performed at time *t.* If no activity is being performed at *t*, these values are set to those of the last activity performed before *t*. For each attribute, there is a condition that must try to satisfy the schedules of the machines $M_1, ...,M_k$.

Given a machine *M* and an activity *a*, each condition associated to *M* is evaluated at time *t,* as a function of the value of the corresponding attribute of *M* at time *t,* and the value of the same attribute of *a.* For example, for the attribute *ink line*, (corresponding to the printing process) the condition is *to preserve the ink line*. If the activity *a* uses machine *M* and is scheduled at starting time *t,* the condition *to preserve the ink line* holds at time *t*, if the value of the attribute *ink line* for *M* at time *t* is equal to the value of the attribute *ink line* of activity *a*. In the practical application, the verification of this condition represents the fact that activity *a* and the previous one use the same ink line.

The problem is to assign each activity to a starting time and to a specific machine trying to verify the conditions. This problem can be considered as a multi-objective combinatorial (MOCO) problems where the objectives are determined by the conditions. In the bibliography which we have found referring to MOCO problems, the multi-objective functions are evaluated after finding a solution. See for entries (Teghem *et al*. 2000, 2001) specifically.

In our problem, the objectives to be fulfilled have a very peculiar characteristic: The conditions (i.e. *to preserve ink line*, etc.) that must be verified, are associated with pairs of activities scheduled consecutively in one machine; whereas (Teghem *et al*. 2000, 2001) need all the activities to be scheduled in order to evaluate the objective functions.

As a result, our algorithm can evaluate the objectives in each step that leads to a solution, as opposed to evaluating the multi-objective function after the whole solution was found, as it is done in (Teghem *et al*. 2000, 2001).

Several schedules could be built, in such a way that activities are scheduled without using the conservation functions and then, a posteriori, a schedule that maximize the objective function could be found. In other words, the conditions can be evaluated a posteriori once the schedule is already built, but an unnecessarily large number of schedules with a low number of conservations that do not lead to a good solution will be generated.

If the criterion to choose an activity depend on conditions that are associated with pairs of activities scheduled consecutively in one machine, it is more convenient to choose an activity evaluating the conditions during the building of the schedule.

Furthermore, if several activities verify different conditions, the algorithm uses the weights to choose the activity to be scheduled. It could be done once the scheduling was already built, at the cost of building an unnecessarily large number of schedules generated without evaluating the conditions during the building of the schedules. Due to this fact, the problem presented here is not a classical MOCO problem.

A comparison of these approaches would be deceptive since we take advantage of particular features of our problem that allows us to guide our search for solutions, whereas the other approaches are much more general. The problem has been initially modeled in (Ibañez *et al*., 2001), using alternative resource sets (Ilog, 1999).

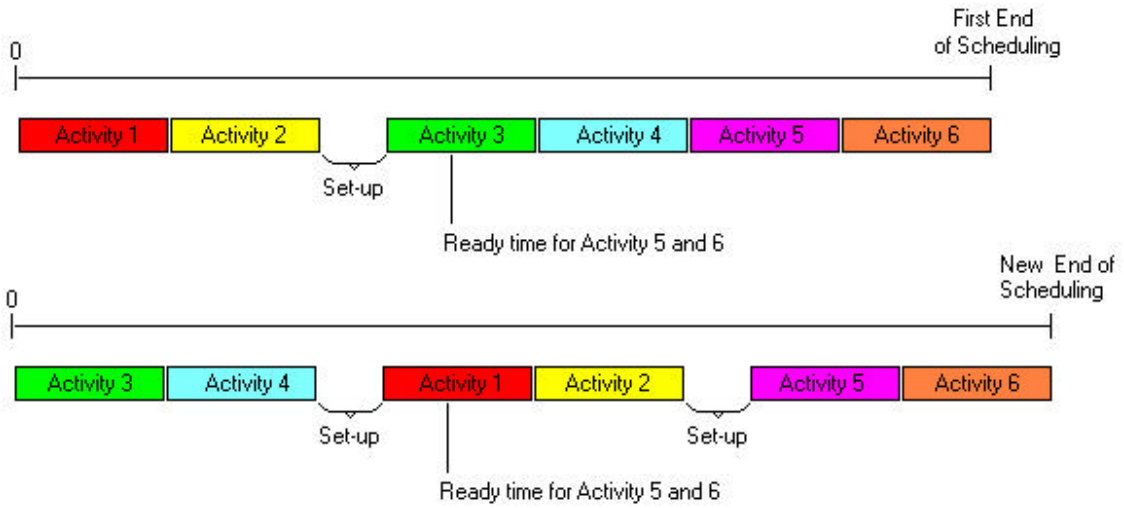
From now on alternative resource sets will be referred to as AltResSets. An AltResSet is a compound resource that contains two or more equivalent resources, called alternative resources, to which activities can be assigned. An AltResSet is defined for each process. Each AltResSet represents a set of machines such as $\{M_1, ...,M_k\}$ and contains *k* alternative resources that represent the machines $M_1, ...,M_k$.

An important requirement of the company is to try to avoid due date violations. An approach to decrease the quantity of due date violations was presented in (Ibañez *et al*., 2002).

In this context, it is not always easy to determine the criterion to choose the best schedule. For instance, consider the following assumptions:

The activities *a1* and *a2* have ink line *il1*, activities *a3*, *a4*, *a5* and *a6* have ink line *il2.* A Set Up time is needed whenever ink lines are changed. Furthermore, the activities a3 and a4 are urgent, and the activities a5 and a6 have an overdue ready time (an overdue delivery of raw material which imposes a lower bound to start).

Consider the following schedules:



Note that a5 and a6 cannot be scheduled immediately after a3 and a4 because of the read time.

Which schedule is the best?

Normally, it depends on many factors like for example cost associated to penalties for due date violations, cost involved in changing ink lines, repercussion of the Set Up operations in the use of the resources and also on the current situation of the company. For instance, if we know that there will be a low demand in the near future, it could be better to chose the second schedule in order to avoid due date violations. On the contrary, if a high demand is foreseen in the near future, the second schedule will avoid due date violations for the time being, but the resource is forced to be available later (since two Set Ups were included) and therefore it is quite likely that new due date violations will arise. As a result, the first schedule can be more satisfactory.

This example is not a classical machine scheduling problems with set-up time and objective defined as a function of Makespan and Lateness. The cost associated to changing ink lines has to be taken into account too. In the problem presented in this paper many, conditions can be associated to each AltResSet and the conditions are different for each AltResSet. In some printing machines, changes do not generate set up time but they do involve additional costs.

Besides, an AltResSet can be formed by many alternative resources with associated functions that express the convenience of assigning a particular alternative resource to the activity.

If we generalize these concepts, we face essentially two competitive criteria. One is to minimize the number of due date violations and the other is to verify as many conditions as possible.

The first criterion is basically customer satisfaction oriented, whereas the second one is usually more convenient for the company. So, there has to be a trade-of f between them in order to adapt the algorithm to the particular situation.

2. Solving the Problem

We will use classical notations of scheduling theory to denote the usual concepts and the notation used in (Ilog, 1999) for more specific concepts. The number of jobs is denoted by *n.* The number of AltResSets is 4. o_{ij} refers to the operation, of job *j* on AltResSet *i* and p_{ij} denotes the duration of o_{ij} ($1 \le i \le 4$ and $1 \le j \le n$). In order to take into account the due dates, we define two attributes associated to the activities: *PriorityWeight* and *MaxEnd*. Each job *j* has a due date, referred as d_j. The values of the attribute *MaxEnd* are set by executing the following pre-processing:

For each job *j* ($1 \le j \le n$)
{Let o_{1j}, o_{2j}, o_{3j} and o_{4j} be the activities (referred to as operations in many books) belonging to the job *j* (Printing, Laminating, Cutting and Packing, respectively)
o_{4j}.MaxEnd = d_j
for i = 3 down to 1 { o_{ij}.MaxEnd = $o_{i+1\ j}$.MaxEnd – $p_{i+1\ j}$}
}

For each activity o_{ij} ($1 \le i \le 4$ and $1 \le j \le n$), $o_{ij}.MaxEnd$ represent the maximum time in which the activity o_{ij} can finish. This value does not change during the execution of the algorithm, whereas $o_{ij}.PriorityWeight$ is initially set to 0 and it increases its value every time that $o_{ij}.End > o_{ij}.MaxEnd$ during the execution of the algorithm ($o_{ij}.End$ represents the end of the activity o_{ij}). It has been assumed that each activity requires only one *AltResSet*.

Let *AltResSets*, *AltResources*, and *Conditions* represent: all the *AltResSets*, all the alternative resources, and all the conditions, respectively. Below we included the functions involved in the algorithms.

StartMin: uses one not scheduled activity as its parameter, and returns the minimal possible start time.

AltResSet: uses an activity as its parameter, and returns the AltResSet required by this activity.

Verify: use an activity *act*, an alternative resource *altRes*, and a condition *cond*, as its parameters, and returns 1 if *act* verifies the condition *cond* at the time *StartMin(act)* with respect to the alternative resource *altRes.* Otherwise the function returns 0.

Conds: uses an AltResSet as its parameter, and returns the set of conditions associated with the argument.

Possible: takes as arguments, an activity *act*, and an alternative resource *altRes*, and returns 1 if it is possible to assign *altRes* to *act* at the time *StartMin(act)*. Otherwise it returns 0.

Weight: Takes a condition and returns a value that represents the degree of importance of that condition.

AltRes: takes an AltResSet and returns the set of alternative resources that are part of the AltResSet.

AltResPreference: uses an activity and an alternative resource as its parameters, and returns a non negative integer number, whose value is set according to the convenience of assigning the alternative resource to the activity.

Given, an activity o_{ij} ($1 \le i \le 4$ and $1 \le j \le n$), an AltResSet *altResSet*, an Alternative Resource $altRes \in AltRes(altResSet)$, and $conds = Conds(AltResSet)$, the functions *AltConvenience, AltResSetConvenience* and *ActivityConvenience* are defined as follows:

$$AltConvenience(o_{ij}, altRes, conds) = Possible(o_{ij}, altRes) * (AltResPreference(o_{ij}, altRes) + \sum_{c \in conds} Verify(o_{ij}, altRes, c) * Weight(c) + o_{ij}.PriorityWeight)$$

$$AltResSetConvenience(o_{ij}, altResSet) = \max_{recAlt \in AltRes(altResSet)} AltConvenience(o_{ij}, altRes, Conds(altResSet))$$

$$ActivityConvenience(o_{ij}) = AltResSetConvenience(o_{ij}, AltResSet(o_{ij}))$$

2.1. Obtaining a Solution

The next algorithm produces a solution in which the number of due date violations depends on the value of the attribute *PriorityWeight* assigned to each activity. *Activities* represent the set of all the activities that have to be scheduled.

repeat
Min = $\min_{1 \le i \le 4,\ 1 \le j \le n}$ StartMin(o_{ij})
(Get the minimum time in which it is possible to schedule an activity).
MinSet = { o_{ij} ($1 \le i \le 4$ and $1 \le j \le n$): StartMin(o_{ij}) = Min}
(Get the set of activities with minimum start time *Min*)
MaxConvenience = $\max_{act \in MinSet}$ ActivityConvenience(act)
Pairs = {(a, altRes) : a∈ MinSet, r = AltResSet(a), altRes∈ AltRes(r),
conds = Conds(r), AltConvenience(a, altRes, conds) = MaxConvenience }
(Get the set of pairs Activity-AlternativeResource that maximize the function *AltConvenience*).
Select an element of the set Pairs. Let's say (*a, altRes*).
Schedule the activity *a* at time *Min* assigning the alternative resource *altRes*.
until All the activities o_{ij} ($1 \le i \le 4$ and $1 \le j \le n$) are scheduled

Algorithm 1. Algorithm to obtain a solution

2.2 Reducing Due Date Violations

The algorithm is based on repeatedly solving the scheduling while trying to verify as many conditions as possible (initially completely disregarding due dates) and calculating the lateness of the activities with respect to the maximum times in which the activities can finish. This information is used in the algorithm in the following iterations so that the delayed activities tend to be scheduled earlier. *k* represents the maximum quantity of iterations.

It is necessary to formalize the objective functions that represent both, the verification of conditions and the fulfillment of due dates. The former represents the convenience of the Company and the latter represent the degree of the customer satisfaction.

In order to calculate the fulfillment of conditions, we need to define the following functions:

ActsAltRes(altRes) : takes as argument an alternative resource, and returns the set of the activities that were scheduled on this alternative resource after the Algorithm 1 is executed.

#ActsAltRes(altRes): takes as argument an alternative resource, and returns the cardinality of the set
ActsAltRes(altRes) (the number of the activities that were scheduled on this alternative resource after the Algorithm 1 was executed).

The number of activities that were scheduled on an alternative resource set *altResSet* is calculated adding up the number of activities scheduled on the alternative resources which belong to *altResSet.*

Formally:

$$\#ActsAltResSet\ (altResSet) = \sum_{altRes \in AltRes(altResSet)} \#ActsAltRes(altRes)$$

VerifyAltRes(*altRes, c*): takes as arguments an alternative resource *altRes* and a condition *c* and returns the number of activities that were scheduled on *altRes* that verify the condition *c*. Formally:

$$VerifyAltRes(altRes, c) = \sum_{a \in ActsAltRes(altRes)} verify(a, altRes, c)$$

VerifyAltResSet (*altResSet, c*): takes as arguments an alternative resource set *altResSet* and a condition *c* and returns the number of activities that were scheduled on *altResSet* that verify the condition *c*.

Formally:

$$VerifyAltResSet(altResSet, c) = \sum_{altRes \in AltRes(altResSet)} VerifyAltRes(altRes, c)$$

MaxFulfillments(*altResSet, c*): takes as arguments an alternative resource set *altResSet* and a condition *c* and returns the maximum number of activities that can be scheduled on *altResSet* verifying the condition *c.*

To understand how the values of these functions are obtained, assume that *altResSet* is formed by only one alternative resource *altRes*, and condition *c* represent ink line. Furthermore, assume that activities *a1*, *a2*, *a3*, *a4* and *a5* with ink lines *il1*, *il1*, *il2*, *il2* and *il1* respectively, turned out to be scheduled on *altRes* after the Algorithm 1 was executed.

For the previous assumptions, we have:

$$\#ActsAltResSet\ (altResSet) = \#ActsAltRes(altRes) = 5,$$
$$VerifyAltResSet(altResSet, c) = VerifyAltRes(altRes, c) = 2 \text{ and}$$

MaxFulfillments(altResSet, c) = 3.

MinFulfillments(*altResSet, c*): takes as arguments an alternative resource set *altResSet* and a condition *c* and returns the minimum number of activities that can be scheduled on *altResSet* verifying the condition *c.*

FulfillmentPercentage(altResSet, c): takes as arguments an alternative resource set *altResSet* and a condition *c* and returns the percentage of activities that were scheduled on *altResSet* verifying the condition *c.* Formally:

FulfillmentPercentage(altResSet, c) = 1 if MaxFulfillments(*altResSet, c*) ≠ MinFulfillments(*altResSet, c*)

Otherwise

FulfillmentPercentage(altResSet, c) =

(VerifyAltResSet(*altResSet, c*) - MinFulfillments(*altResSet, c*)) /
(MaxFulfillments(*altResSet, c*) - MinFulfillments(*altResSet, c*))

So, *FulfillmentPercentage(altResSet, c)* varies from 0 to 1 (0 if *VerifyAltResSet(altResSet, c)* = *MinFulfillments(altResSet, c)* and 1 if *VerifyAltResSet(altResSet, c)* = *MaxFulfillments(altResSet, c)*).

FulfillmentPercentageAltResSet(altResSet) =

$\sum_{c \in Conds(altResSet)}$ *FulfillmentPercentage(altResSet, c) * Weight(c) /* $\sum_{c \in Conds(altResSet)}$ *Weight(c)*

Now we can define the percentage of fulfillment of conditions considering all of the alternative resource sets.

ConditionFulfillmentPercentage =

$\sum_{altResSet \in AltResSets}$ *FulfillmentPercentage*AltResSet*(altResSet, c) * Weight(c) / 4*

So, *ConditionFulfillmentPercentage* varies from 0 to 1 since *FulfillmentPercentage(altResSet, c)* does.

Now, let's include some definitions to calculate the quantities of orders delivered-on-time. Let *Jobs* be the set of jobs which represent all of the orders of the customers, *#Jobs* the total number of jobs and C_j the time in which the job *j* finishes after the *Algorithm1* is executed.

L_j represents the lateness of the job *j* and *#DueDatesViolations* the quantity of due date violations. Formally:

$L_j = C_j - d_j$ and

#DueDatesViolations is defined to be the cardinality of the set $\{j \in Jobs : L_j > 0\}$

Now we can define the percentage of orders delivered-on-time as follows:
OnTimeOrdersPercentage = (#Jobs - #DueDatesViolations) / #Jobs

So, *OnTimeOrdersPercentage* varies from 0 to 1 depending on the quantities of due date violations.

Finally, if *ConditionFulfillmentWeight* represents the weight assigned to fulfillment of the conditions and *OnTimeOrdersWeight* the weight assigned to orders delivered-on-time, the objective function that will be used in the algorithm is defined as follows:

ObjectiveFunction = OnTimeOrdersPercentage OnTimeOrdersWeight +*
*ConditionFulfillmentPercentage *ConditionFulfillmentWeight*
(with *OnTimeOrdersWeight + ConditionFulfillmentWeight = 1*)

```
iter = 0;
bestObjectiveFunction = 0;
for each i
    for each j
        { o_ij.PriorityWeight = 0}      (initially due dates will be disregarded)
repeat
  execute Algorithm 1
  if ObjectiveFunction > bestObjectiveFunction
          then
            {bestObjectiveFunction = ObjectiveFunction;
              store:
                the schedule produced by Algorithm 1,
                OnTimeOrdersPercentage,
                ConditionFulfillmentPercentage
            }
   for each i
    for each j
      {
       L_ij  = o_ij.End – o_ij.MaxEnd
       (note that L_ij is a concept different from L_j, associated to an activity)
       if L_ij > 0 then o_ij.PriorityWeight = o_ij.PriorityWeight + L_ij *Step
      }
  iter = iter + 1
until (L_ij ≤ 0 for all 1 ≤ i ≤ m, 1 ≤ j ≤ n) or (iter > k)
```

Algorithm 2. Algorithm to obtain a solution minimizing due dates violation

The greater the lateness is for an activity the greater its priority to be chosen will be in the next iteration. *Step* determines how fast the delayed activities will increase their priorities

The value of Step has to be carefully chosen. An inadequate value for Step can produce bad results. There are two cases.

a.- In each iteration, the weights and the preferences of the alternative resources compete with the lateness of activities. If we choose too high a value for Step, we take the risk that the weights and the preferences of the alternative resources

have no influence whatsoever. In this case, the algorithm will first schedule all the activities with lateness, however in a blind manner.

b.- Conversely, if the value of Step is too low, the lateness will exert insignificant influence and the scheduling will mainly be driven by the weights and the preferences of the alternative resources.

So the performance of the algorithm is strongly dependent on the value chosen for Step.

3. Results

The current implementation provides a very detailed output which includes the percentage of fulfillment of the conditions associated to all of the alternative resource sets, the quantity of due date violations, as well as the sum and the average of the due date violations in terms of time.

We will include a very small set of results that will suffice to show the main issues. In the following table we include the output obtained from 3 different sources of data (Batch 1, Batch 2 and Batch 3).

As the data are taken from the real application, they include many conditions to be fulfilled and involve several resources. As a result, the data include too many details that would complicate the overall understanding, so we include and explain only the output produced by the program.

On Time Orders Weight	*ConditionFulfillmen Weight*	OnTime Orders Percentage	Condition Fulfillment Percentage	Objective Function
0	1	0.32	0.82	0.82
0.7	0.3	0.62	0.69	0.669
1	0	0.91	0.63	0.91

Table1: Percentage of On-Time Orders & Fulfillment of conditions and Objective Function for Batch 1

OnTime Orders Weight	*ConditionFulfillmen Weight*	OnTime Orders Percentage	Condition Fulfillment Percentage	Objective Function
0	1	0.27	0.79	0.79
0.7	0.3	0.58	0.67	0.607
1	0	0.60	0.56	0.60

Table2: Percentage of On-Time Orders & Fulfillment of conditions and Objective Function for Batch 2

OnTime Orders Weight	*ConditionFulfillmen Weight*	OnTime Orders Percentage	Condition Fulfillment Percentage	Objective Function
0	1	0.21	0.76	0.76
0.7	0.3	0.54	0.59	0.555
1	0	0.78	0.54	0.78

Table3: Percentage of On-Time Orders & Fulfillment of conditions and Objective Function for Batch 3

It can be seen that as we increase the weight for orders delivered-on-time, the On-Time Orders Percentage increases and the Condition Fulfillment Percentage diminishes, which is normally the desired result. Given the values for *OnTime OrdersWeight and ConditionFulfillmentWeight*, the program finds a schedule that maximizes the Objective Function, established as a function of these weights. The objective function is used by the algorithm to find a scheduling that is suitable for the given weights and it must not be used to choose the weights. The election of the adequate weight must be done as a function of the necessities of the company. Depending on many variables mentioned earlier (costs, situation of the company, etc.) the company must choose appropriate weights to produce the desired schedule.

In spite of the current factors that could lead to the company to choose one particular set of Weights, there are also some practical issues that should be taken into account. If the company needs to give priority to the orders delivered-on-time, it could make sense to choose *OnTimeOrdersWeight = 1.*

However, if we are dealing with Batch 2, we can see that there is just a slight difference between the *OnTimeOrdersPercentage* for the last two rows (0.58 if *OnTimeOrdersWeight = 0.7* and 0.60 if *OnTimeOrdersWeight = 1*) but the difference between *ConditionFulfillmentPercentage* is significant (0.67 for the 2nd row and 0.56 for the 3rd row). As a result, if you choose the 2nd row, you get a significant advantage in terms of fulfillment of conditions, paying a small price in terms of due date violations.

4. Conclusion

In this work, an algorithm for solving a Scheduling for Flexible Package Production which allows assignment weights to set an appropriate trade-off between due date violation reduction and fulfillment of conditions was analyzed.

The setting of these weights has to be carefully chosen according to the desired output, considering costs, situation of the company, and so on. The performance of the algorithm is strongly dependent on the value chosen for Step.

Although the algorithm presented in (Ibañez *et al*., 2001) solves the problem of fulfillment of conditions and (Ibañez *et al*., 2002) reduces the number of due date violations, the situation of some companies requires reaching a solution in which a trade-off between these requirements is desired. The present work also allows to try with different weights to analyze the output and to select the adequate values according to the desired wanted schedule, driven by the current condition of the company.

Even though the results obtained up to now with the algorithm presented here are acceptable for the companies which we are working with, an exhaustive evaluation has to be done on a large variety of data and this is the task that is being carried out at the present moment.

References

Ibañez F., D. Diaz and R. Forradellas (2001). Scheduling for flexible package production. Proceedings IEPM'2001. Quebec, Canada.

Ibañez F., D. Diaz and R. Forradellas (2002). Scheduling for Flexible Package Production Minimizing Due Times Violations. Proceedings Eighth International Workshop on Project Management and Scheduling - PMS2002. Valencia, Spain.

Ilog (1999). Ilog Schedule Reference Manual Version 4.4. Ilog, France.

Teghem J., D. Tuyttens and E. L. Ulungu (2000). An interactive heuristic method for multiobjective combinatorial optimization. Computers and Operations Research , Vol. 27. 621-634.

Teghem J., P. Fortemps, D. Tuyttens and T. Loukil (2001). Solving multi-objective production scheduling problems using metaheuristics. Proceedings IEPM'2001. Quebec, Canada.