A Weight-Based Algorithm to deal with Due Dates in Flexible Package Production Scheduling

Francisco Ibáñez, Germán Zavalla, Daniel Díaz, Raymundo Forradellas
LISI – Laboratorio Integrado de Sistemas Inteligentes
Instituto de Informática – Dpto. de Informática
Universidad Nacional de San Juan
{fibanez, ddiaz, kike}@iinfo.unsj.edu.ar

Abstract

This paper is an extension of a previous paper published in [2]. 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 perform 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 [1]. As mentioned there, some due dates could not be fulfilled on time. An approach to decrease the quantity of due dates violations was presented in [2]. Roughly speaking, the algorithm presented in [1] is entirely dedicated to verify as many conditions as possible disregarding due dates violations. By the other hand, the algorithm shown in [2] was focussed to reduce the number of due dates violations paying the price of decreasing the fulfilment of conditions. Roughly speaking, the first approach favour the company whereas the second one is more convenient for the customers.
The present work includes an algorithm which allows to assign weights to set an appropriate trade-off between due date violations reduction and fulfillment of conditions.

Keywords: Applications, Scheduling Problems, Constraints Satisfaction, Production, Flexible Packaging.

1. Introduction

This paper is an extension of a previous paper published in [2]. The problem consist on scheduling several jobs, which involve four process: Printing, Laminating, Cutting and Packing. 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 starting at 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 the activity *a*. In the practical application, the verification of this condition represents the fact that the 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 that we have found about MOCO problems, the multi-objective functions are evaluated after finding a solution (see for example [4] & [5]). 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. The nature of these conditions allows the algorithm to 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 the other approaches. 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 [1], using alternative resource sets [3]

From now on alternative resource sets will be referred 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 $M_1, ..., M_k$.

Two competitive objectives are relevant in this problems: to try to verify as many conditions as possible, and to meets due dates.

When conditions are not verified, the factory has a cost. For instance, if the ink line is not preserved, additional set up times for changing the ink line are necessary increasing the waste of ink and degrading the use of the printing machine. By the other hand, if due dates are nor met, penalties have to be usually paid.

Assume that two different schedules are produced. The first one tends to meets due dates, whereas the second one tends to verify as many conditions as possible.

Which schedule should be chosen?

If we know that there will be low demand in the near future, it is normally better to chose the first schedule. By doing so, more due dates are met and the consequent bad use of machines do not seriously affect the future performance since the demand in the near future is low. On the contrary, if a high demand for the near future is foreseen, the first schedule will avoid due date violations for the time being, but the resources are forced to be available later (bad use of machines due to Set Ups) and therefore it is quite likely that new due date violations will arise in the near future. As a result, the second schedule is more appropriate.

Generalizing this concepts, we are facing basically 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.

The present work includes an algorithm which allows to assign weights to set an appropriate trade-off between due date violations reduction and fulfillment of conditions.

In [2], no objective function is used. Instead, different measures to evaluate the quality of the results are provide. In this paper, an objective function that depends on the assigned weights is included and the algorithm chooses the schedule that maximize the objective function. In this way, the output depends on the assigned weights, that are set to meets more due dates (customer oriented) or to verify more conditions (company oriented).

2. Solving the problem

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 *dueDate(J)*. The values of the attribute *MaxEnd* are set by executing the following pre-processing:

```
For each job $J$
  {Let $a_1, a_2, a_3$ and $a_4$ be the activities belonging to the job $J$  (Printing,
   Laminating, Cutting and Packing, respectively)
    $a_4$.MaxEnd = dueDate(J)
   for i = 3 down to 1  {$a_i$.MaxEnd = $a_{i+1}$.MaxEnd – duration($a_{i+1}$) }
  }
```

For each activity *a*, *a.MaxEnd* represent the maximum time in which the activity *a* can finish. This value does not change during the execution of the algorithm, whereas *a.PriorityWeight* is initially set to 0 and it increases its value every time that *a.End* > *a.MaxEnd* in the reached solution (*a.End* represents the end of the activity *a*). 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: takes as argument an activity not scheduled, and returns the minimal possible start time.

AltResSet: takes as argument an activity, and returns the AltResSet required by this activity.

Verify: takes as arguments an activity *act*, an alternative resource *altRest*, and a condition *cond*, and returns 1 if *act* verify the condition *cond* at the time *StartMin(act)* with respect to the alternative resource *altRest.* Otherwise the function returns 0.

Conds: takes as argument an AltResSet, 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: takes an activity and an alternative resource, 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 *a*, an AltResSet *altResSet*, an Alternative Resource $altRes \in AltRes(altResSet)$, and $conds = Conds(AltResSet)$, the functions *AltConvenience, AltResSetConvenience* and *ActivityConvenience* are defined as follows:

$$
\begin{aligned}
&AltConvenience(a, altRes , conds) = \\
&\quad Possible(a, altRes) * (AltResPreference(a, altRes) \\
&\quad + \textstyle\sum_{c \in conds} Verify(a, altRes,c)*Weight(c)) + a.PriorityWeight) \\
&AltResSetConvenience(act, altResSet) =
\end{aligned}
$$

$$Max_{altRes \in AltRes(altResSet)} AltConvenience(act, altRes, Conds(altResSet))$$
$$ActivityConvenience(act) = AltResSetConvenience(act, AltResSet(act))$$

2.1. Obtaining a solution

The next algorithm produces a solution in which the number of due dates violation depend 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_{act∈Activities} StartMin(act)
    (Get the minimum time in which it is possible to schedule an activity).
    MinSet = {act∈Activities : StartMin(act) = Min}
    (Get the set of activities with minimum start time Min)
    MaxConvenience = Max_{act∈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 maximise the function AltConvenience).
    Select (randomly) 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 are scheduled
```

Algorithm 1. Algorithm to obtain a solution

2.2 Reducing due dates violation.

The next 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. *n* represent the maximum quantity of iterations.

For each activity *a* that requires the AltResSet *r*, such that *a.Lateness* is greater than zero, *a.PriorityWeight* is calculated taking into account the lateness of *a*, the maximum lateness of the activities that require *r*, the weights and preferences associated to *r*, the preferences of using one or another alternative resource of *r*, and the number of the current iteration.

Given, an activity *a*, an AltResSet *altResSet*, and an Alternative Resource $ar \in AltRes(altResSet)$, we define the following functions in order to calculate the value of *a.PriorityWeight* if *a.Lateness* is greater than zero.

$RequiredActivities(altResSet) = \{a \in Activities : AltResSet(a) = altResSet\}$
$MaxWeight(altResSet) = \sum_{c \in Conds(altResSet)} Weight(c)$
$MaxAltResPreference(altResSet) =$
$\qquad Max_{a \in RequiredActivities(altResSet),\ ar \in AltRes(altResSet)} AltResPreference(a, ar)$
$MaxWeightPlusPref(altResSet) = MaxWeight(altResSet) +$
$\qquad MaxAltResPreference(altResSet);$
$MaxLateness(altResSet) = Max_{a \in RequiredActivities(altResSet)} (a.End - a.MaxEnd)$
($a.End - a.MaxEnd$ represents the Lateness of activity *a*)

It is necessary to formalize the objective functions that represent both, the fulfillment of conditions and the delivered-on-time orders. The former represent the convenience of the Company and the later represent the degree of satisfaction of the customers.
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.

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:
$\text{VerifyAltRes}(altRes,\ c) = \sum_{a \in ActsRes(altResSet)} verify(a,\ altRes,\ c)$

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

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.*

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.*

FulfillmentPerc(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:

FulfillmentPerc(altResSet, ~~c) =~~
$$(\text{VerifyAltResSet}(altResSet,\ c) - \text{MinFulfillments}(altResSet,\ c)\ )\ /$$
$$(\text{MaxFulfillments}(altResSet,\ c) - \text{MinFulfillments}(altResSet,\ c)\ )$$

So, *FulfillmentPerc(altResSet, c)* varies from 0 to 1 (0 if *VerifyAltResSet(altResSet, c) = MinFulfillments(altResSet, c)* and 1 if *VerifyAltResSet(altResSet, c) = MaxFulfillments(altResSet, c)*).
Now we can define the percentage of fulfillment of conditions considering all of the alternative resource sets.

$$ConditionFulfillmentPerc = \sum_{altResSet \in AltReSets} \sum_{c \in Conds(altResSet)} FulfillmentPerc(altResSet, c) * Weight(c) \;/\; \sum_{altResSet \in AltReSets} \sum_{c \in Conds(altResSet)} Weight(c)$$

So, *ConditionFulfillmentPerc* varies from 0 to 1 since *FulfillmentPerc(altResSet, c)* does.
Now, let's include some definitions to calculate the quantities of delivered-on-time orders.
Let's *Jobs* be the set of jobs which represent all of the orders of the customers, *#Jobs* the total number of jobs and *end(J)* the time in which the job *J* finishes after the *Algorithm1* is executed.
Lateness(J) represents the lateness of the job *J* and *#DueDatesViolations* the quantity of due dates violations. More formally:
Lateness(J) = end(J) – dueDate(J) and
#DueDatesViolations is defined to be the cardinality of the set $\{J \in Jobs : Lateness(J) > 0\}$
Now we can define the delivered on-time-order percentage as follows:

OnTimeOrdersPercentage = (#Jobs - #DueDatesViolations) / #Jobs

So, *OnTimeOrdersPercentage* varies from 0 to 1 depending on the quantities of due dates violations.
Finally, if *ConditionFulfillmentWeight* represents the weight assigned to fulfillment of the conditions and

OnTimeOrdersWeight the weight assigned to delivered on-time-orders, the objective function that will be used in the algorithm is defined as follows:

OnTimeOrdersWeight and *ConditionFulfillmentWeight* are assigned values within 0 and 1.

$$ObjectiveFunction = OnTimeOrdersPercentage * OnTimeOrdersWeight + ConditionFulfillmentPerc * ConditionFulfillmentWeight$$

(with $OnTimeOrdersWeight + ConditionFulfillmentWeight = 1$)

The *Algorithm 2* works as follows. As a consequence of the first line, *Algorithm 1* is initially executed disregarding due dates. The solution initially found is dedicated to verify as many conditions as possible.

The algorithm then iterates *n* times or stops if no lateness is found. In each iteration, after executing the *Algorithm 1*, values for *a.Lateness* are determined and the values of *a.PriorityWeight* are evaluated for each activity *a* in order to be used in the next iteration.

The value of *n* has to be high enough to produce good results as will be explained later on.

```
iter = 0;
bestObjectiveFunction = 0;
for each a∈Activities  {a.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,
                ConditionFulfillmentPerc
            //(Only the last values are kept)
          }

  //updates a.PriorityWeight for all activity
  for each r∈AltResSets
   {
    maxLateness = MaxLateness(r);
    for each a∈RequiredActivities(r)
     {
       a.Lateness = a.End – a.MaxEnd;
       if (a.Lateness > 0)
          then
            a.PriorityWeight =  (iter/n) * MaxWeightPlusPref (r) *
                                (1 + a.Lateness /maxLateness)
          else
            a.PriorityWeight =0;
     };
   };
  iter = iter + 1
until (a.lateness <= 0 for all a∈Activities) or (iter > n)
restore:
    The schedule produced by Algorithm 1,
    OnTimeOrdersPercentage,
    ConditionFulfillmentPerc
```

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

3. Obtained 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 input data 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).

		Batch 1		Batch 2		Batch 3	
OnTime Orders Weight	*Cond Fulfillm Weight*	OnTime Orders Percent	Condition Fulfillment Percentage	OnTime Orders Percent	Cond Fulfill Perc	OnTime Orders Percent	Cond Fulfill Perc
0	1	0.35	0.81	0.28	0.78	0.24	0.75
0.7	0.3	0.63	0.68	0.59	0.68	0.57	0.60
1	0	0.92	0.64	0.61	0.55	0.81	0.53

Table1: Percentage of On-Time Orders and Fulfillment of conditions

It can be seen that as we increase the weight for delivered-on-time orders, the On-Time Orders Percentage increases and the Condition Fulfillment Percentage diminishes. Depending on many factors 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 delivered-on-time order, it could make sense to choose *OnTimeOrdersWeight = 1* (the 3rd row in the Table 1). However, if we are dealing with the Batch 2, we can see that there is just a slight difference between the *OnTimeOrdersPercentage* for the last two rows (0.59 if *OnTimeOrdersWeight = 0.7* and 0.61 if *OnTimeOrdersWeight = 1*) but the difference between *ConditionFulfillmentPerc* is significant (0.68 for the 2nd row and 0.55 for the 3rd row). As a result, if you choose the 2nd row, you get a significant advantage in terms of fulfillment of condition, paying a small price in terms of due dates violations.

4. Conclusion

In this work, an algorithm for solving a Scheduling for Flexible Package Production minimizing Due Times violations has been examined. This paper presents an algorithm that improved the results generated in [2] for some particular cases. That is, mainly, cases in which there are many conditions associated with the resources and also the weights of the resources are very different among them. Typically, the performance of the algorithm improves as the number of iterations grows, but of course the execution time increases as well.

Although the results obtained up to now with the algorithm presented here are better than those obtained in [2] for the mentioned cases, an exhaustive evaluation on both algorithms has to done on a large variety of data and this is the task that is being carried out at the present moment. MODIFICA EL ALG. ANTERIOR [2].....

In this work, an algorithm for solving a Scheduling for Flexible Package Production which allows to assign weights to set an appropriate trade of between due date violations reduction and fulfillment of conditions.

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

Although the algorithm presented in [1] solve the problem of fulfillment of conditions and the [2] reduces the number of due date violations, the situation of some companies requires to reach a solution in which a trade of between these requirements is desired. The present works allows also to try different weights and to analyze the output and to select the adequate values according to wanted schedule driven by the current situation of the company.

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

References

[1] Ibañez F., Diaz D., Forradellas R.,"Scheduling for flexible package production", Proceedings IEPM'2001. Vol. 1, 385-400, Quebec, Canada, 2001. Selected work

for the International Journal of Production Economics (IJPE) topic "Operation Management", http://www.elsevier.com
[2] Ibañez F., Diaz D., Forradellas R.," An Algorithm for Minimising Due Times Violations in Flexible Package Production ", CACIC 2002, Bs As, 2002.
[3] "Ilog Schedule- Reference Manual Version 4.4", Ilog, France, 1999.
[4] Teghem J., Tuyttens D., Ulungu E.L., "An interactive heuristic method for multiobjective combinatorial optimization". Computers and Operations Research , Vol. 27. 621-634, 2000.
[5] Teghem J., Ph. Fortemps, Tuyttens D., T. Loukil "Solving multi-objective production scheduling problems using metaheuristics", Proceedings IEPM'2001. Vol. 1, 385-400, 2001.