A Constraint Programming Approach to FMS Scheduling. Consideration of Storage and Transportation Resources

Luis J. Zeballos and Gabriela P. Henning

INTEC (Universidad Nacional del Litoral - CONICET)
Güemes 3450 (3000)
Santa Fe, Argentina
{zeballos, ghenning}@intec.unl.edu.ar

Abstract

Flexible manufacturing systems (FMS) require efficient scheduling strategies to achieve their principal advantage of being able to combine high flexibility with high productivity. This paper presents a novel constraint programming (CP) formulation addressing the FMS scheduling problem for systems that work under the operational policy of part movement. The formulation addresses some critical features found in these problems and proposes a search strategy to speed/up the solution process. This work also presents an ad-hoc procedure, based on different initialisation points, that aims at making problems tractable when their size is big enough to make difficult the attainment of good quality solutions in a reasonable CPU time.

Keywords: Constraint satisfaction, Flexible manufacturing systems, Scheduling, Resource Limitations.

1. Introduction

Flexible manufacturing systems (FMS) require efficient scheduling strategies to achieve their principal advantage of being able to combine high flexibility with high productivity [Liu & MacCarthy 97]. In most FMSs there are two operational policies [Gamilla & Motavalli 03]. One is the tool movement policy, under which parts are assigned to machines and the necessary tools are transported to the various machines to perform the required operations. Thus, a tool transportation system is required. The other one is the part movement policy, under which the parts are transferred from one machine to another since a given machining center does not have all the required tools to process a part. Therefore, a part's handling device is necessary.

Due to their inherent complexity, a wide variety of solution techniques have been employed to tackle the various scheduling problems that can be found in several industrial domains. Thus, researchers and practitioners have resorted to mathematical programming and heuristic approaches, network algorithms, dynamic programming, intelligent agent methodologies, genetic algorithms, etc. Constraint Programming (CP) [Marriot & Stuckey 99] is one of the techniques that has gained increased attention in the last years and that has been successfully applied to the scheduling domain [Alfonso & Barber 00]. In spite of that, very few contributions have made use of CP in FMS scheduling problems [El Khayat et al. 03]. Moreover, these contributions are very preliminary.

This paper presents a novel CP formulation addressing a part movement class of FMS scheduling problem. It involves two components. The CP model itself, consisting of the set of constraints that represents the problem being tackled, and a search procedure. The proposed search strategy takes advantage of some domain features that improve the performance of the standard domain reduction and constraint propagation algorithms included in the adopted CP language. The proposed formulation is based on the

OPL language, supported by ILOG Solver [ILOG03a], and employs some specific scheduling constructs available in ILOG Scheduler [ILOG03b].

The CP model addresses some important features found in industrial problems, such as equipment ready-times as well as release times and due-dates of jobs. But, more importantly, it successfully takes into account constraints on resources such as storage capacity (machine buffers) and material transport devices, which represent critical aspects of practical problems. It is worth to remark that the proposed formulation does not decouple two essential decisions associated to the problem: (i) the assignment of job operations to machines (i.e. loading) and (ii) the sequencing of the assigned operations. Thus, the FMS scheduling problem is tackled in a global way, proving better quality solutions than the widely advocated "loading then sequencing" method.

Regarding the search strategy, the paper proposes distinct approaches for different size problems. For smaller ones, the aim is to start the search process from a good quality solution that exhibits a balanced machine load. For big size ones, an ad-hoc algorithm, based on different initialisation points, is proposed. This domain specific procedure aims at making problems tractable by speeding-up the solution process.

The contribution includes computational results for test problems reported in the open literature [Liu & MacCarthy 97]. Moreover, problems of different sizes, that consider a variety of objective functions (such as minimizing the maximum completion time, mean completion time and tardiness measures), are tackled. The obtained results show both, very good quality and a remarkable computational efficiency. The rest of this paper is organized as follows. Section 2 characterizes the problem under study and Section 3 introduces the CP model. In this section, the adopted nomenclature is presented, the basic problem constraints and objective functions are introduced. Section 4 shows the implemented search strategy and the proposed iterative procedure to find satisfactory solutions for big size problems. To illustrate the application of the proposed CP model, Section 5 presents computational results and discusses experiences associated to various case studies. Finally, section 6 reports conclusions and future work.

2. Problem Definition

This work considers a flexible manufacturing cell with a fixed number of machines, a single material handling device and a given number of jobs or parts that require several processing operations. In order to build the CP model, the following assumptions are taken into account:

- All parts require the same number and sequence of operations. The operations' precedence is fixed and known in advance.
- Processing times do not depend on the alternative machines.
- Pre-processed parts entering the system and finished parts leaving the system are stored in different buffers having infinite capacity. In turn, in-process parts are stored in machines' input buffers having limited capacity.
- Processing times, order release-times and due-dates, machine ready-times, part transportation times and sizes of local buffers are all fixed and known.
- The transport device cannot be used as a temporary storage buffer.
- Transportation times are equal for all types of travelling movements.
- When the transport device goes from the buffer for pre-processed parts to the buffer for finished parts, or vice versa, it must cross the area where machines are located. Then, it can be assumed that the transportation time taken by the transport device to go from one of these buffers to the other is equal to the time taken by the transport device to go from the buffer for pre-processed (or finished) parts to the machine's zone, plus the time taken by the transport device to go from the machine's zone to the buffer for finished (or pre-processed) parts.

3. Constraint Programming Model

The proposed model will be gradually introduced; starting with the nomenclature and a basic formulation, that will later incorporate additional features. Then, constraints for handling both, parts transportation and machine buffers will be presented. Finally, different objective functions will be posed.

3.1. Nomenclature

Subscripts

j, k	Parts or jobs to be processed.
p, p'	Operations associated to parts, that need to be scheduled
u	Machines to process the required operations.

Sets

U	Set of machines and their associated input buffers.
J	Set of parts.
P	Set of operations.
s	Set of alternative machines.

Model Variables

Task-Timing Related Variables: In most scheduling approaches, tasks to be scheduled (activities to be done) and resources are two essential modeling elements. ILOG OPL Studio and ILOG Scheduler have incorporated them as pre-defined variables. In this contribution, *Task*, *TaskSt* and *TaskTr* are three types of activities to be included in the model. The first one represents processing tasks associated to parts (machining operations). Thus, activity $Task_{jp}$ models the execution of operation p on part j. The second one represents storage tasks that take place in buffers and the third one represents part handling tasks that are executed by the transport device. All of the activities are characterized by means of duration, starting time and ending time variables (e.g. *Task.duration*, *Task.start* and *Task.end*), that are related among themselves.

The activity type *TaskSt* has two kinds of tasks associated to it:

$TaskStIni_j$	corresponds to the storage of part j previous to the execution of the first required operation on part j. It occurs in the buffer of the machine in which the first operation is done.
$TaskStInter_{jpp'}$	is related to the storage task of part j performed between two successive operations (p and p'). It takes place in the buffer of the machine to which the second operation is assigned.

In the same way, the activity type *TaskTr* originates three different kinds of tasks:

$TaskTrIni_j$	related to the transportation activity of part j from the storage place for pre-processed parts to the machine that will execute the first operation.
$TaskTrInter_{jpp'}$	corresponds to the transportation of part j between two different processing units where operations p and p' are respectively carried out.
$TaskTrFin_j$	is related to the transportation activity of part j from the machine that has executed the last operation on the part to the storage place for finished parts.

In addition, the model handles the following extra variables.

MK	Maximum completion time or Makespan.
T_j	Tardiness of order j with respect to its promised due-date.

Parameters

d_j	Due date of part j
t_{jp}	Processing time of operation p on part j.
rt_j	Release time of part j.
tp	Transportation time between two machines.
rd_u	Ready time of machine u.
sb_u	Capacity of machine u's buffer.

In turn, the model handles the following special parameters.

mach	ILOG parameter indicating that each processing unit $mach_u$ is a unary resource (cannot be shared by two activities at a given time) that can execute just one operation on a part at any time. Unary resources having similar capabilities can be grouped into resource sets; this is the case of the machines $mach_u$, which are grouped into the s set.
c	ILOG parameter indicating that the handling device c is a unary resource and that can execute just one transportation task on a part at any time.
buffer	ILOG parameter indicating that each storage buffer $buffer_u$ is a discrete resource (one having a fixed capacity). In this case, it is called sb and represents the maximum amount of parts that can be stored in the resource at a given time.

3.2. Model Characteristics

The CP model has been written in the OPL constraint programming language, supported by ILOG Solver, employing some specific scheduling constructs available in the ILOG Scheduler package. One of them is ***requires*** which, as shown in Eq. (1), enforces the assignment of resources (on the right hand side) demanded by activities (on the left hand side). Another special construct is ***precedes***, which has the purpose of imposing a proper sequence of non-overlapping activities. As shown in Eq. (2), the activity located at the right hand side cannot be initiated until the activity on the left hand side has been finished. One of the most important special constructs is ***ActivityHasSelectedResource***, which easily allows handling alternative resources. It acts like a predicate that evaluates to true (assumes a value equal to one) when an activity has been assigned to a specific resource belonging to a set of alternative resources.

3.3. Model Constraints and Performance Measures

Assignment, Timing, Sequencing and Precedence constraints

$$Task_{jp} \; requires \; s \qquad \forall j \in J, \forall p \in P \tag{1}$$

$$Task_{jp} \; precedes \; Task_{jp'} \qquad \forall j \in J, \forall p, p' \in P, ord(p') = ord(p) + 1 \tag{2}$$

$$Task_{jp}start \geq rt_j + tp \qquad \forall j \in J, \forall p \in P, ord(p) = first(P) \tag{3}$$

$$activityHasSelectedResource(Task_{jp}, s, mach_u) \Rightarrow Task_{jp}start \geq rd_u \qquad \forall j \in J, \forall p \in P, ord(p) = first(P), \forall u \in U \tag{4}$$

Constraint (1) is an assignment relation prescribing that each machining operation *p* associated to a job *j* must be assigned to just one processing unit $mach_u$ belonging to the set of alternative resources *s*. Constraint (2) enforces a proper sequencing of all the consecutive machining operations (*p* and *p'*) that need to be performed on each part *j*. Eqs. (3)-(4) place lower bounds on the start time of the first operation *p* executed on part *j*. Eq. (3) sets a bound due to the release time of job *j* (rt_j) plus the time required to move such part from the buffer for pre-processed parts to the machine executing the first operation (*tp*). Eq. (4) fixes another bound that depends on the ready time of the machine assigned to the first operation (rd_u). In this case *ActivityHas SelectedResource* evaluates to true when $Task_{jp}$ has been assigned to $mach_u$, which belongs to the set of alternative units *s*.

Transportation constraints

$$TaskTrIni_j.duration = 2 * tp \qquad \forall j \in J \tag{5}$$

$$TaskTrFin_j.duration = 2 * tp \qquad \forall j \in J \tag{6}$$

Constraints (5)-(6) set the duration of the initial and final transportation tasks, respectively. Eq. (5) prescribes that the duration of part *j*'s initial movement is two times the basic transportation time. It includes the time taken by the unloaded handling device to first go from the processing unit where it is currently located to the buffer for pre-processed parts where *j* is placed and then, from such buffer to the one of the machine that will execute the first operation on *j*. In the case of Eq. (6), it includes the time taken by the transport device to go from the processing unit executing the last operation on *j* to the finished parts' buffer and from such buffer to any machine that would require it.

$$\begin{gathered} activityHasSelectedResource(Task_{jp}, s, mach_u) \neq \\ activityHasSelectedResource(Task_{jp'}, s, mach_u) \Rightarrow \\ TaskTrInt_{jpp'}.start = Task_{jp}.end \;\; \& \\ TaskTrInt_{jpp'}.end \leq Task_{jp'}.start \;\; \& \\ TaskTrInt_{jpp'}.duration = tp \quad \forall j \in J, \forall p, p' \in P \\ ord(p') = ord(p) + 1, ord(p) < last(P), \forall u \in U \end{gathered} \tag{7}$$

$$\begin{gathered} activityHasSelectedResource(Task_{jp}, s, mach_u) = 1 \; \& \\ activityHasSelectedResource(Task_{jp'}, s, mach_u) = 1 \Rightarrow \\ TaskTrInt_{jpp'}.start = Task_{jp}.end \;\; \& \\ TaskTrInt_{jpp'}.end \leq Task_{jp'}.start \;\; \& \\ TaskTrInt_{jpp'}.duration = 0 \quad \forall j \in J, \forall p, p' \in P \\ ord(p') = ord(p) + 1, ord(p) < last(P), \forall u \in U \end{gathered} \tag{8}$$

Constraint (7) indicates that when two consecutive operations (*p* and *p'*) associated to a given job are processed in different units, an intermediate transportation task takes place and its duration assumes a positive value. In such a case, the $TaskTrInt_{jpp'}$ activity should start when the execution of operation *p* on part *j* has finished and should end when the next processing operation *p'*, is about to start. On the other hand, as indicated by constraint (8), when two consecutive operations belonging to a given job are executed in the same unit, there is no need for an intermediate transportation activity and its duration is set to null.

$$TaskTrIni_j \; requires \; c \quad \forall j \in J \tag{9}$$

$$TaskTrFin_j \; requires \; c \quad \forall j \in J \tag{10}$$

$$\begin{gathered} TaskTrInt_{jpp'}.duration > 0 \Leftrightarrow \\ TaskTrInt_{jpp'} \; requires \; c \quad \forall j \in J \\ \forall p, p' \in P, \; ord(p') = ord(p) + 1, ord(p) < last(P). \end{gathered} \tag{11}$$

Eqs. (9) - (11) enforce the assignment of the three kinds of transportation tasks to the handling device *c* carrying out the movements of parts. The definitive assignment of $TaskTrInt_{jpp'}$ depends on its duration. Indeed, this kind of activity is actually assigned if and only if its duration is greater than zero; otherwise, the activity does not exist and there is no need to assign it to the transport device.

$$TaskTrFin_j.start = Task_{jp}.end \qquad \forall j \in J, \forall p \in P, ord(p) = last(P) \tag{12}$$

Constraint (12) prescribes that the final movement of a given part *j* should start just at the time the last operation performed on it has finished.

$$\begin{gathered}TaskTrInt_{jpp'}.duration > 0 \ \& \\ TaskTrInt_{kqq'}.duration > 0 \ \& \\ activityHasSelectedResource(Task_{jp}, s, mach_u) \neq \\ activityHasSelectedResource(Task_{kq'}, s, mach_u) \ \& \\ TaskTrInt_{kqq'} \ precedes \ TaskTrInt_{jpp'} \Rightarrow \\ TaskTrInt_{kqq'}.end + tp \leq TaskTrInt_{jpp'}.start \\ \forall\ j,k \in J,\ j \neq k,\ \forall\ p,p',q,q' \in P, \\ ord(p') = ord(p)+1, ord(q') = ord(q)+1, \\ ord(p) < last(P), ord(q) < last(P)\ ,\ \forall\ u \in U\end{gathered} \tag{13}$$

$$\begin{gathered}TaskTrInt_{kpp'}.duration > 0 \ \& \\ activityHasSelectedResource(Task_{jq}, s, mach_u) \neq \\ activityHasSelectedResource(Task_{kp'}, s, mach_u) \ \& \\ TaskTrInt_{kpp'} \ precedes \ TaskTrFin_j \Rightarrow \\ TaskTrInt_{kpp'}.end + tp \leq TaskTrFin_j.start \\ \forall\ j,k \in J,\ j \neq k,\ \forall\ p,p',q \in P, ord(q) = last(P), \\ ord(p') = ord(p)+1, ord(p) < last(P), \forall\ u \in U\end{gathered} \tag{14}$$

Constraints (13) - (15) prescribe the temporal relationships existing between various transportation activities carried out by the same handling device on two different parts. Constraint (13) represents the case in which two actual intermediate movements are demanded to transport parts *j* and *k*. When the intermediate transportation activity corresponding to part *k* is performed before the intermediate transportation activity corresponding to part *j*, a *tp* time is needed by the transport device to go from the machine where the part *k* is unloaded to another machine where the part *j* is to be picked-up in order to be transported. On the other hand, constraint (14) prescribes the temporal relationship between an actual intermediate transportation activity and a final one. When the intermediate transportation activity is performed before the final one, a *tp* time is needed by the material handling device to go from the machine where part *k* is to be unloaded (to execute the *p'* operation) to the unit where part *j* is to be picked-up to be transported to the finished parts' buffer. This constraint does not prescribe the situation in which the final transportation activity takes place before the intermediate one. In this case, a *tp* time is also needed by the transport device to go from the finished parts' buffer, where part *j* is unloaded to a machine where part *k* is to be loaded (to be moved to another unit). However, provided the definition of the final transportation task given in eq. (6), this time is included in the duration of the activity and therefore this situation does not need to be taken into account in the expression.

$$\begin{gathered}TaskTrInt_{kpp'}.duration > 0 \ \& \\ activityHasSelectedResource(Task_{jq}, s, mach_u) \neq \\ activityHasSelectedResource(Task_{kp}, s, mach_u) \ \& \\ TaskTrIni_j \ precedes \ TaskTrInt_{kpp'} \Rightarrow \\ TaskTrIni_j.end + tp \leq TaskTrInt_{kpp'}.start \\ \forall\ j,k \in J,\ j \neq k,\ \forall\ p,p',q \in P, ord(q) = first(P), \\ ord(p') = ord(p)+1, ord(p) < last(P), \forall\ u \in U\end{gathered} \tag{15}$$

Constraint (15) models the temporal relationship associated to the case in which the initial transportation activity takes place before the intermediate one. Its explanation and underlying rationale is analogous to the one of constraint (14).

Storage constraints

$$\begin{gathered}TaskStIni_j \ requires \\ (activityHasSelectedResource(Task_{jp}, s, mach_u)) \ buffer_u \\ \forall\ j \in J,\ \forall\ p \in P, ord(p) = first(P), \forall\ u \in U\end{gathered} \tag{16}$$

$$\begin{gathered}TaskStInt_{jpp'} \ requires \\ (activityHasSelectedResource(Task_{jp'}, s, mach_u)) \ buffer_u \\ \forall\ j \in J,\ \forall\ p, p' \in P, ord(p') = ord(p)+1, \\ ord(p) < last(P), \forall\ u \in U\end{gathered} \tag{17}$$

Eqs. (16)-(17) enforce the assignment of storage tasks to the proper machine input buffers, depending on the assignment of operations to machines. Thus, when a given operation on a certain part is assigned to machine *mach$_u$*, the corresponding storage activity must also be assigned to this machine's buffer (*buffer$_u$*). This applies to both types of storage tasks.

$$TaskStIni_j.start = TaskTrIni_j.end \quad \forall\ j \in J \tag{18}$$

$$\begin{gathered}TaskStIni_j.end = Task_{jp}.start \\ \forall\ j \in J, \forall p \in P, ord(p) = First(P)\end{gathered} \tag{19}$$

Constraints (18)-(19) prescribe the starting and ending times of initial storage activities. Eq. (18) indicates that any initial storage activity must start when its associated initial transportation task ends, and Eq. (19) sets that it finishes when the first processing operation performed on the part begins.

$$\begin{gathered}TaskStInt_{jpp'}.start = TaskTrInt_{jpp'}.end \quad \forall\ j \in J, \\ \forall\ p, p' \in P, ord(p') = ord(p)+1, ord(p) < last(P)\end{gathered} \tag{20}$$

$$\begin{gathered}TaskStInt_{jpp'}.end = Task_{jp'}.start \quad \forall\ j \in J, \\ ord(p') = ord(p)+1, ord(p) < last(P)\end{gathered} \tag{21}$$

In turn, Eqs. (20)-(21) model the starting and ending times of intermediate storage activities. Eq. (20) states that any intermediate storage activity must start when its associated intermediate transportation task ends, Eq. (21) asserts that it ends when the corresponding operation on the part starts.

Performance measures' constraints

Makespan

$$Task_{jp} \ \textit{precedes} \ MK \qquad (22)$$

$$\forall j \in J, \forall\, p \in P, ord(p) = last(P)$$

$$Min\ MK \qquad (23)$$

Mean Completion Time

$$Min \sum_{j \in J} Task_{jp}.end \Big/ np \qquad (24)$$

$$\forall\ p \in P, ord(p) = last(P), np = card(J)$$

Tardiness (25)

$$T_j \geq Max\,(0, Task_{jp}.end - d_j)$$

$$\forall\ j \in J, \forall\ p \in P, ord(p) = last(P)$$

$$Min \sum_{j \in J} T_j \qquad (26)$$

The CP model can deal with several objective functions such as makespan *MK*, mean completion time and performance indexes related to job tardiness T_j. If makespan is chosen as the problem objective function, constraints (22) and (23) should also be included in the model since (22) defines the makespan concept. Constraint (24), which computes the mean completion time, becomes part of the model if this objective function is adopted. Finally, Eq. (25) defines the tardiness concept by assigning a positive value to variable T_j if the finishing time of the last operation on job *j* is greater than the job's due-date d_j. This equation should be included in the model if the total tardiness defined by Eq. (26) becomes the problem performance measure.

4. Search Strategies

Constraint programming systems provide default search procedures; however, most of them allow to easily program user defined search strategies. This type of support is fundamental for hard combinatorial optimisation problems demanding special search procedures [Van Hentenryck 99]. This functionality, represents one of the most important characteristics of constraint programming languages, which can lead to improvements in performance by implementing search strategies tailored to particular domains. Ad-hoc search strategies may explore the entire search space (total strategies) or a portion of it (partial strategies), as in those cases in which the values that model variables can adopt when instantiated are restricted and only a portion of the search space is actually explored.

4.1. Search Strategy for Small Size Problems

A global domain-specific search strategy, that specifies the mechanism by which operations are assigned to machines, has been defined for low dimensionality problems. Conceptually, this strategy starts from an initial balanced assignment of operations to machines, avoiding uneven loads. Moreover, provided there is an operations' precedence, the search strategy pursues the assignment of operations to machines according to such ordering. This preliminary assignment allows to obtain a good quality solution in a short time as well as to have a good upper bound that permits to prune poor quality solutions. The initial assignment of operations to machines is done in sequential order, starting from the first operation p_1 executed on each part, following with the second one and continuing in the same way with the remaining ones. To achieve this ordered trial of assignments, the procedure ***C_Constraint_Builder*** (see Fig. 1), assembles a list, named *C*, of length *np*nj*, containing as elements several primitive constraints of the form $c_i = (Task_{jp} \mapsto x)$. This list acts as a constraint during the search process. At the very beginning, all assignments belonging to *C* are free $(Task_{jp} \mapsto x)$; but then, at any point of the global search process, *C* will keep track of the already fixed variables.

```
Let nu ← card(U)
Let nj ← card(J)
Let np ← card(P)
Let C ← primitive constraints' list: $c_1..c_n$
Let $c_1, c_2, ......, c_n$ ← primitive constraints of
  the form $Task_{jp} \mapsto x$ for non-assigned
  tasks or $Task_{jp} \mapsto u$ for tasks assigned to
  machines
Let n = np*nj, number of primitive
  constraints
Let x ← variable denoting a machine to be
  assigned
Let PartList ← list of parts
Let OperList ← list of operations
Let MachList ← list of machines

C_Constraint_Builder ( )
 Let i = 1
 for(p in OperList) do
  for (j in PartList) do
   append  $c_i = (Task_{jp} \mapsto x)$ to C
   i = i + 1
  endfor
 endfor
 return C

MachWorkList_Builder( )
  u = first element of MachList
  for (p  in OperList)  do
    for (j in PartList)  do
      Let $MachWorkList_{jp}$ ← Empty list
      for (k  in 1…nu) do
        append  u  to $MachWorkList_{jp}$
        if (u = last element of MachList)
        then
          u = first element of MachList
        else
          u = next element to u in MachList
        endif
      endfor
      if (u = last element of MachList)
      then
        u = first element of MachList
      else
        u = next element to u in MachList
      endif
    endfor
  endfor
  return MachWorkList_jp
```

Figure 1. Conceptual definition of auxiliary procedures employed during the search strategy

For example, let us consider a problem in which four parts, requiring three operations each one, are to be scheduled in three machines. If at a given point during the exploration of the search space, only the first operation *p1* executed on parts *j1* and *j2*, has been tentatively allocated to machines u_1 and u_2, and all other assignments have not been attempted yet, the corresponding C constraint, having 12 primitive constraint elements, will adopt the following format:

$\{c_1 =(Task_{j1p1} \Rsh u_1),\ c_2 = (Task_{j2p1} \Rsh u_2),\ c_3 = (Task_{j3p1} \Rsh x),\ c_4 = (Task_{j1p2} \Rsh x),\ c_5 = (Task_{j2p2} \Rsh x),\ \ldots\ldots,\ c_{12} = Task_{j4p3})\}$

As seen, the order in which tasks appear in C reflects the sequence in which the assignment of tasks to machines will be tried out during the search process. Therefore, each task $Task_{jp}$ represents a node of the search space in which various machine assignments will be explored. In order to pursue a balanced machine load, the order in which machines are attempted to be assigned at each $Task_{jp}$ node, will be different. Thus, each node has a particular list of assignable machines, referred as $MachWorkList_{jp}$. The sequence of elements in such list represents the order in which machines would try to be assigned to $Task_{jp}$ during the search process. The set of machine lists is created by the ***MachWorkList_Builder*** procedure shown in Fig. 1.

Referring to the previous example, the lists of machines associated to the first three tasks are: $Task_{j1p1} \mapsfrom \{u_1,u_2,u_3\}$, $Task_{j2p1} \mapsfrom \{u_2,u_3,u_1\}$; $Task_{j3p1} \mapsfrom \{u_3,u_1,u_2\}$. Thus, at the start of the search process the first three tasks in C are allocated to three different machines, thus initiating a balanced assignment.

The code that formalizes the proposed domain specific search strategy is presented in Figure 2.

```
Let c_1 ,c_2 ,……,c_n primitive constraints of
  the form Task_jp ↱ x for non-assigned
  tasks or Task_jp ↱ u for tasks assigned to
  machines
Let C and C1 ← constraints
Let x ← variable representing a machine
  to be assigned
Search_Strategy(C)
  If (all tasks have just been assigned)
  then
    return Partial_Satisfiable(C)
  else
    choose c_i first not assigned task
    (Task_jp ↱ x) in C
    Let MachworkList_jp ← List of machines
    associated to Task_jp
    for u in MachworkList_jp do
      let C1 be obtained from C by
      instantiating x with u:
      c_i = (Task_jp ↱ u)
      if Partial_Satisfiable(C1) then
        if Search_Strategy(C1) then
          return true
        endif
      else c_i = (Task_jp ↱ x)
      endif
    endfor
    return false
  endif
```

Figure 2. Search strategy for small problems

Search_Strategy is a recursive procedure that resorts to another one that verifies if assignments are feasible. Such procedure, named ***Partial_Satisfiable*** (see Fig. 3), operates on constraint C testing whether each one of the primitive constraints in which x has already been instantiated is satisfiable.

```
Partial_Satisfiable (C)
  for i in 1..n do
    if the task in c_i is assigned then
      if satisfiable(c_i) = false then
         return false
      endif
    endif
  endfor
  return true
```

Figure 3. Constraint feasibility test

When the test that is being applied to a given c_i primitive constraint does not succeed, the ***Partial_Satisfiable*** procedure returns `false` and consequently, the ***Search_Strategy*** procedure disengages the last assignment (See Fig. 2) that was done in relation to the $Task_{jp}$ corresponding task. Thus, it starts a backtracking step, under which a new u machine belonging to $MachworkList_{jp}$ would be tried. If all the machines belonging to this list have been tried and have also failed, then, the backtracking step would be more profound and would affect the task associated to the previously treated primitive c_{i-1}

constraint. Therefore, if necessary, all the assignments can be revised and the backtracking process can unwrap up to the very first machine allocation that was initially performed.

4.2. Search Strategy for Large Problems

When problems are of bigger size, an iterative strategy is proposed to obtain good quality solutions in a reasonable CPU time. This strategy consists in solving the problem several times, as many as the number of parts associated to the problem being solved, but each time applying a different partial search process. The most important change among the processes is the starting point; thus, each one begins from a different balanced assignment of tasks to machines. Each partial search process belonging to the strategy begins from a distinct starting point and stays searching during a limited time; when the time limit is over the best obtained solution is taken and added to the model as an upper bound to be applied on the next partial search procedure. This process is repeated until the last partial procedure reaches its time limit.
The rationale behind this approach is based on the fact that for each partial process, solutions are quickly obtained. However, after a short time no more solutions are found in a fast fashion and the next solution may be obtained in an uncertain time (order of several minutes, hours or days, depending on numerical aspects and problem sizes). Each partial search is conceptually similar to the one implemented for small problems. In fact, the first search procedure is identical to the one applied in the previous section. The remaining ones change in the order in which tasks $Task_{jp}$ appear in the primitive constraints belonging to `C`. Provided such distinct ordering of tasks in `C`, the search space will be differently explored by ***`Search_Strategy(C)`***.

It should be remarked that the policy of first assigning the first operation to be executed on each part, then the second one, and so on, is to be kept. Thus, in order to generate a different sequencing of tasks in `C`, a different arrangement of parts in `PartList` is just necessary. If the ordering of elements in this list changes, then, the output of the ***`C_Constraint_Builder`*** procedure will be different. So, each partial search procedure will be associated to a different ordering of parts, that will be referred as `InitWorkList`. Fig. 4 shows the procedure that generates the various part orderings associated to the different partial search processes.

```
Let nj ← card(J)

Let PartList ← list of parts

InitWorkList_Builder()
  j = first element of PartList
  for  (l in 1 to nj)  do
    Let InitWorkList_l ← empty list
    for (j  in PartList)  do
      append  j  to InitWorkList_l
      if (j = last element of PartList)
      then
        (j = first element of PartList)
      else
        (j = next element to j in PartList)
      endif
    endfor
    j = next element to j in PartList
  endfor
  return  InitWorkList
```

Figure 4. Conceptual definition of the initial working list for each partial search

5. Examples and Computational Results

In order to examine the effectiveness of the proposed CP model and of the corresponding search strategies, a set of test problem instances of different sizes have been generated in a random fashion. Also, a small problem reported by [Liu & MacCarthy 97] has been considered, too. Small problems comprise case-studies having a range of 2 to 4 machines, 3 to 5 jobs, 1 to 4 operations per job (where any machine can be selected to process any operation), and machine storage buffers having capacities that range from 0 to 2. They correspond to the first five data sets included in Table 1. In large problems, in which the strategy introduced in section 4.2 was applied, the number of machines was increased. It ranges from 5 to 7, as well as the number of jobs, that ranges from 6 to 10. However, all the other problem characteristics were kept the same. They correspond to the last three data sets located at the bottom of Table 1.

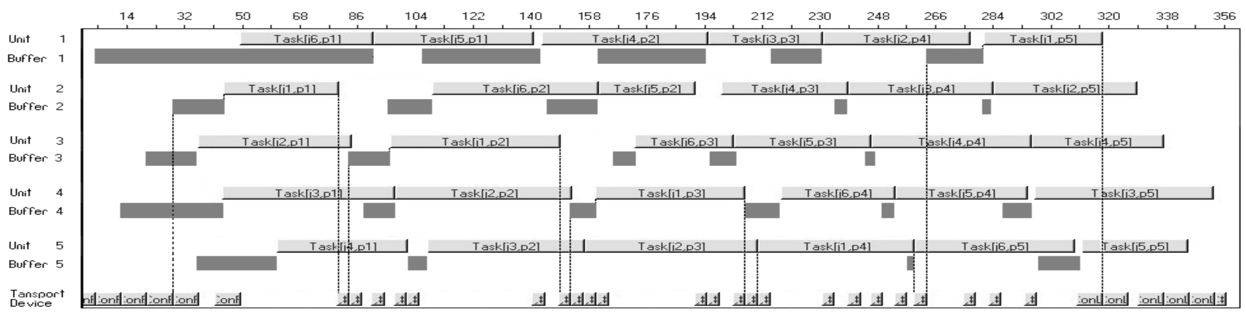

Figure 5. Gantt diagram corresponding to a case study having six jobs, five operations, five machines and a buffer capacity of one part. Makespan (1) was adopted as the scheduling criterion

Data Set	Objective Function	Objective Value		CPU Time	
		CP	MILP	CP	MILP
J=4, *P*=2, *U*=2, *sb*=0	1	187	187	0.36	0.59
	2	133.2	133.2	0.08	0.42
	3	30.8	30.8	0.02	0.26
J=5, *P*=2, *U*=3, *sb*=1	1	171	171	1.03	173.25
	2	143.4	143.4	38.11	110.77
	3	17	17	1.02	1.06
J=5, *P*=3, *U*=3, *sb*=1	1	247	248*	97.06	2000
	2	204.20	204.2	238.63	2000
	3	271	321*	53.51	2000
J=3, *P*=4, *U*=3, *sb*=1	1	217	217	0.02	16.34
	2	199.33	199.33	0.03	222.72
	3	24	24	0.03	7.84
J=5, *P*=3, *U*=4, *sb*=2	1	189	192*	60.26	2000
	2	171.4	174.4*	83.14	2000
	3	299	307*	30.67	2000
J=6, *P*=5, *U*=5, *sb*=1	1	332*	490*	2000	2000
	2	330*	384*	2000	2000
	3	196*	---	2000	2000
J=5, *P*=7, *U*=5, *sb*=1	1	367	---	0.609	2000
	2	340.4	---	5.017	2000
	3	357	---	7.22	2000
J=8, *P*=5, *U*=5, *sb*=1	1	428*	---	2000	2000
	2	386.87*	---	2000	2000
	3	1297*	---	2000	2000

* Suboptimal solution; --- No solution was found.

Table 3. Computational results corresponding to several examples

Table 1 includes results obtained with the proposed approach as well as results of the mixed-integer linear programming (MILP) model introduced by [Liu & MacCarthy 97]. The adopted scheduling criteria are makespan (1), mean completion time (2) and tardiness (3). As seen, the computational effort demanded by the CP approach is not affected by the choice of the objective function. The computation was carried out on a Pentium VI, 1.8 Ghz, PC with 1Gbyte of RAM memory.

The time limit to obtain solutions was set to 2000 seconds of CPU time. Then, to obtain comparative results between the CP and MILP approaches, the time limit used for the large problems' search strategy was computed as the previous one divided by the number of particular searches to be performed. From Table 1, it can be observed that in all the cases CP results are better than the MILP approach ones. In addition, while the problem sizes are of reduced dimensionality, the times of both solution methodologies are similar, but when the problem sizes increase, the CP model is the only one that produces good solutions in small CPU times. Figure 5 depicts the Gantt diagram corresponding to one of the large problems (Data set 6: J=6, P=5, U=5, sb=1). For illustrative purposes, dotted lines help tracing the path associated to part *j1*.

6. Conclusions

A novel CP formulation that addresses a class of FMS scheduling problems has been proposed. It can handle many critical features usually found in industrial problems, such as equipment ready-times, release times and due-dates of jobs, machine buffers of limited capacity, transportation activities carried out by a material handling device, etc. The approach includes a CP model as well as domain specific search strategies. Regarding the model, it should be remarked that it does not decouple two essential decisions associated to the problem: (i) the assignment of job operations to machines and (ii) the sequencing of the assigned operations. Therefore, the problem can be tackled in a global way. Thus, better quality solutions than the widely advocated “loading then sequencing” heuristic method can be obtained. The approach has been tested with a variety of case studies and has rendered excellent solutions, requiring very little computational effort. In the case of small and medium size examples optimal solutions have been reached. For large problems, an ad-hoc partial search strategy has been introduced. It allows reaching suboptimal solutions in reasonable CPU times.

Future work includes improving the search strategy by making a more intelligent choice of the search procedure’s starting points. Moreover, a more clever assignment of operations to machines will be attempted. In addition, the sequencing of the transportation activities will be studied in detail.

Acknowledgments

Authors acknowledge financial support from ANPCyT under Grant 11-14717, from CONICET (PEI 6191) and “Universidad Nacional del Litoral” (CAI+D 2002, 5-30).

References

[Alfonso & Barber 00] M.I. Alfonso and F. Barber. “Combinación de Procesos de Clausura y CSP para la Resolución de Problemas de “Scheduling”. Revista Iberoamericana de Inteligencia Artificial, 9. pp 20-26. (2000).

[El Khayat et al. 03] G. El Khayat, A. Langevin and D. Riopel. “Integrated Production and Material Handling Scheduling using Mathematical Programming and Constraint Programming” Proceedings of CPAIOR’03. Montreal, Canada, May 8-10. (2003).

[Gamilla & Motavalli 03] M.A. Gamila and S. Motavalli. “A Modeling Technique Loading and Scheduling Problems in FMS”. Robotics and Computer Integrated Manufacturing, 19. pp. 45-54. (2003).

[ILOG 03a] ILOG OPL Studio 3.7. User’s Manual, France. (2003).

[ILOG 03b] ILOG Scheduler 6.0. User’s Manual, France. (2003).

[Liu & MacCarthy 97] J. Liu and B.L. MacCarthy. “Theory and Methodology. A Global MILP Model for FMS Scheduling”. European Jounal of Operations Research, 100. pp 441-453. (1997).

[Marriot & Stuckey 99] K. Marriot and P. Stuckey. “Programming with Constraints. An Introduction”. The MIT Press, Cambridge, Massachusetts. (1999).

[Van Hentenryck 99] P. Van Hentenryck. “The OPL Optimization Programming Language”. The MIT Press, Cambridge, Massachussets. (1999).