Abstract

This paper presents a study relative to the design of a Decision Support System (DSS) for production scheduling, more precisely for the processing-order release function in a flanging shop of an aircraft company. The principles used rely on the Constraint Analysis which aims at characterizing the set of feasible schedules. A mock-up has been programmed with a Constraint Logic Programming (CLP) language. The synthesis of the Constraint Analysis with CLP is called as a Constraint-Based Approach.

Key-words : Decision Support System, Scheduling, Constraint Analysis, Constraint Logic Programming.

1In Proceedings of PAP’95 Practical Application of Prolog, Paris (France), April 4-7, 1995, pp. 405-422.
INTRODUCTION

In this paper, we present some results from the SCOOP² project supported by the French Ministry for Scientific Research under a contract with the LAAS-CNRS, Dassault Aviation and the European Institute of Cognitive Science and Engineering [Haudot et al. 93]. Within this framework, the emphasis has been placed on scheduling systems operating in little automated workshops producing small runs. In this context, it is almost impossible to take account of the complete implicit expertise of decision makers, as it encompasses pieces of information that are both diverse and widely different. Humans then play a key-role by carrying out specific operations and making decisions whenever disturbances or emergencies occur.

Solving this problem through an optimization approach seemed ill-suited, partly due to the difficulties we have in modelling the various preferences, but also in obtaining the agreement of the operator on a final solution taken without his contribution. We preferred an approach based on decision support, in which the human operator keeps his decision role and controls the solving process, while the system provides him aid facilities for selecting relevant choices and checking their consistency. This principle also seemed realistic since the number of decisions the operator must solve is not very large, compared to the number of constraints he must respect.

We have thus specified a system that integrates interactively the decisions of the operator while providing the possibility to go back to any of them if the current state does not fit any longer his criteria or is not feasible from the constraint satisfaction point of view.

The need of a backtracking search procedure and the major role of the constraints led us to use a Constraint Logic Programming (CLP) tool particularly dedicated to the solving of combinatorial Constraint Satisfaction Problems such as our scheduling problem [Dincbas et al. 88a, 88b]. This kind of languages includes efficient constraint propagation mechanisms that can drastically reduce the search space. Unlike the majority of their applications in the scheduling field, we only take benefit of the constraint propagation mechanisms and not of the generation functions (so-called “labelling”) from which the authors build specific heuristics by automating the choice of the variables and values at each step. This labelling is entirely left to the decision maker, in charge of applying a specific know-how and integrating the dynamic context of the shop-floor.

²SCOOP for "Système COopératif pour l'Ordonnancement de Production" (Co-operative System for Production Scheduling).
Our subject of interest was then the co-operation between two interactive poles, the man and the computer, and how this co-operation can take benefit of an efficient implementation of a constraint-based reasoning in scheduling. This need of co-operation becomes more and more important with the growing development of Group Decision Support Systems or “Groupwares”.

The paper is organized as follows. In Section 1, we define the decision problem and the links between our approach and the techniques used in the framework of Constraint Satisfaction Problems. Section 2 is dedicated to a description prompted by an industrial application, while Section 3 presents the CLP language we employed: CHIP (COSYTEC S.A.). The modelling and the programming are discussed in Section 4. Some results and conclusions are given in Section 5.

1 DECISION PROBLEM IN SCHEDULING

The scope of the SCOOP project concerns shops that deal with problems related to manufacturing. Schematically, a scheduling problem [Baker 74][GOThA 93] involves who should do what and when. It is often an NP-complete problem [Garey & J. 79]. Operational Research attempts to find solutions to these combinatorial problems, most of the time by designing specific heuristics [McCarthy & L. 93]. The result is a schedule that represents a forecasted plan for carrying out the various tasks. It can be used as a guideline for decision making. However, the initial plan can no longer be used facing the disturbances because of its lack of flexibility. Of course, it can be periodically recomputed by taking into account the current state of the production system, but a preliminary characterization of solutions would then be more appropriate. That is the objective of the so-called constraint analysis [Erschler et al. 76, 80, 91]. It is based on a set of deductive rules that reveals some essential properties for all solutions. The process eliminates, as soon and as much as possible, incompatible decisions with respect to the constraints. The domain of certain decision variables can be dramatically reduced before any decision is made about their value. This procedure also enables the detection of some inconsistencies between different types of constraints. Once done, it can be followed by a phase where the user can make decisions according to his expert knowledge [Bel et al. 89]. Given the formulated constraints, this approach favours the system flexibility while guaranteeing for the consistency of the decisions. It avoids calculating systematically a complete solution again when the initial plan is partially disrupted.
Constraint analysis can be viewed as an instantiation of a more general problematic, the **Constraint-Based Approach** for problem solving. This has given rise to theoretical researches, **Constraint Satisfaction Problems** [Tsang 93], and also to the creation of **Constraint Logic Programming** languages [Van Hentenryck 89], e.g., PrologIII, CLP(3), Charme or CHIP. The latter was selected to devise our system; it is presented in Section 3. Combining constraint analysis principles and CLP languages may have some advantages. From the efficiency point of view, a real improvement has been made, because of the dedicated nature of these languages, compared with realizations implemented in general languages such as Pascal, C or ADA. Conversely, some kinds of reasoning suggest certain improvements of the internal mechanisms of constraint management in CLP languages.

### 2 THE APPLICATION

The study case for the SCOOP project is a Dassault-Aviation flanged element manufacturing workshop. This workshop produces small quantities of a highly diversified range of primary parts, *i.e.*, without assembly. The manufacturing process includes part routings in a cutting out centre, various heat treatments, flanging on a hydraulic press and manual finishing. About 1000 parts are produced each week from 3000 types with 12 different operation sequences.

This shop is in charge of a limited series production, which implies a great flexibility in management and the necessity to react on a very short term basis. Moreover production processes have a low level of automation. Routings can be different for each product; thus the workshop operates as a *Job-shop*. Each part is identified by its reference, routing, class and due date.

On the first station, the cutting out one, the problem is to determine the sets of parts which will be cut out simultaneously on each metal sheet. The built sets must comply with various requirements relating to the class, geometric overlapping, filling rate and time constraints associated with the part delivery dates. The installation of the parts on the metal sheets is the overriding factor as it conditions all the processing orders that must be scheduled on the other work stations. It is considered as an *order release*. At present, the sets of parts which are automatically created have to be modified and then

---

3 An instance of the CSP involves a set of variables, a domain for each variable specifying the values to which it may be assigned, and a set of constraints on the variables. The constraints define which combinations of domain values are allowed. One is asked to assign values to variables such that all constraints are simultaneously satisfied [Nuijten & A. 94].
validated by an operator. The problem is that the current system suffers many
drawbacks, particularly as the workshop's real status is not taken into account and
certain pieces of information indispensable for decision-making cannot be displayed.
In parallel to the SCOOP project, a complete schedule system called MADE has been
specified and prototyped for this workshop [Chamard et al. 92, 94][Bellone et al. 95].

3 CHIP

To implement our mock-up, we chose the CHIP (Constraint Handling In Prolog)
language [Dincbas et al. 90]. It is a language based on PROLOG (Edinburgh syntax). A
declarative syntax coupled with a propagation mechanism makes it a tool very powerful
for our problem. To set constraints in an a priori way can reduce variable domains.
This mechanism of implicit propagation of constraints enables the pruning of the search
tree during the resolution. Indeed, as soon as a decision is made (i.e., variable
instantiation), the set of constraints in which it appears is considered, not only to
simplify its expression but also to deduce new restrictions on the domains of the
remaining free variables. This active way of managing constraints (“test-and-generate”)
can be opposed to the classical naïve principle of constraints checking (“generate-and-
test”). Amongst the main reasons which led us to prefer the CHIP language rather than
others, we would point out :

- the strong structural relationship between Constraint Analysis rules and CLP, that
  enables us to re-use some of the previous software packages, written in standard
  Prolog ;

- the specialized nature of CLP languages ;

- the efficiency of constraint propagation mechanisms on domain variables, and
  particularly the possibility of managing discontinuous ranges ;

- the advanced graphical primitives for building interfaces.

Among all the CHIP primitives [COSYTEC 93], we recall hereafter only some of them,
employed in the following sections :

- domain declarations (e.g., X :: 0..100) and algebraic constraints (e.g., #<, #>=, #\=) ;
- domain_info(X,Min,Max,Size,Occurrences,Active): returns various information about domain variable X (smallest, largest values and number of values in the domain of X, number of constraints attached to X, number of unsolved constraints attached to X);

- notin(X,From,To): removes all the values between From and To from the domain of variable X;

- minimum(X,List): enforces that X is the minimum of the elements of a list List;

- maximum(X,List): enforces that X is the maximum of the elements of a list List;

- if Cond then Pred1 else Pred2: If the condition Cond is true for any remaining values in the domains of the variables involved in Cond then the predicate Pred1 is executed. If Cond is false for any remaining values in the domains of these variables then Pred2 is executed. If Cond is still liable to be true or false, nothing is executed and the conditional constraint is delayed (conditional propagation);

- cumulative(Starts,Durations,Ress,Ends,Surfs,High,End,IntVal): expresses the fact that at any moment the total amount of the resources required for the tasks processing does not exceed a given limit.

Moreover CHIP provides an object extension, namely CHIP++. It allows to building better data structures and more readable programming. For instance, to assign value “ValueA” to attribute “attribute1” of an object “ObjectX”, the syntax is: ObjectX@attribute1 <- ValueA.

4 MODELING AND IMPLEMENTATION IN CHIP++

4.1. The geometric problem

For a given metal sheet, parts must be placed in such a way that any two parts do not overlap each other given the total available area of the rectangular sheet. The non-overlapping problem is solved by a constraint programming based 2-D geometry module [Chamard & F. 94]. We will only mention here the main principles of its design:

- the parts are discretized as sets of overlapping rectangles;
- the decision variables are the position and the orientation of a part (three rotations of 90° are possible);

- the solving uses a backtrack procedure which alternates labelling and a conditional propagation of the constraints;

- the labelling method is a *dichotomic labelling* [Dincbas et al. 88c].

The predicate for non-overlapping between two rectangles defined by their positions (x and y represent the coordinates of the lower left corner of the rectangle) and their dimensions (length and width) is given below.

```prolog
non_overlapping(Rectangle1,Rectangle2) :-
    X #= Rectangle1@x, XX #= X + Rectangle1@length,
    Y #= Rectangle1@y, YY #= Y + Rectangle1@width,
    X1 #= Rectangle2@x, XX1 #= X1 + Rectangle2@length,
    Y1 #= Rectangle2@y, YY1 #= Y1 + Rectangle2@width,
    if X #<= XX1 then
        if X1 #<= XX then
            (if Y #<= YY1 then YY #<= Y1,
             if Y1 #<= YY then YY1 #<= Y),
        if Y #<= YY1 then
            if Y1 #<= YY then
                (if X #<= XX1 then XX #<= X1,
                 if X1 #<= XX then XX1 #<= X).
```

In a future version of our application, we will use the *diffn/6* primitive constraint of CHIP which should lead to a substantial saving of memory space.

### 4.2. The temporal problem

Parts are placed on sheets with respect to three types of temporal constraints:

- individual time windows (one for each part) ; (1)

- parts aggregation constraints (one for each sheet) ; (2)
- disjunctive constraints on the cutting machine. (3)

Constraints (1) associate an initial time window [earliest start-time, latest finish-time] to realize the cutting of each part. It is derived from the time window [release date, due date] associated to the corresponding order and the knowledge of the durations of the remaining operations following the cutting out.

Constraints (2) come from some technological features of the cutting machine. For each metal sheet, cutting operations are chained and no part can be available before the whole cutting process of the sheet is achieved. It implies that:

- the aggregated cutting operation on one sheet cannot start before the maximum earliest start-time of its components;

- the aggregated cutting process for one sheet cannot finish after the minimum latest finish-time of its components.

Consequently a time window is associated to each sheet, which is the temporal intersection of individual time windows associated to elementary cutting operations of its parts. This time window must be large enough to include the sum of the cutting durations.

Finally, disjunctive constraints (3) come from the use of a single cutting machine, which implies that aggregated operations have to be sequenced.

4.2. Time constraint propagation

The allocation of parts on sheets are strongly dependent of the temporal constraints. The goal of a co-operative DSS is first to show infeasibilities as soon as the current set of decisions becomes inconsistent with time constraints. Constraint propagation mechanisms can also be used to restrict the possibilities left to the parts not yet placed.

In this paragraph, some rules of constraint analysis for temporal reasoning are presented. These rules are applied to reinforce the decision-problem consistency. In the first part, we propose rules for the consistency of binary constraints (arc-consistency) [Mackworth 77], i.e., rules associated with a pair of tasks. We also consider in the second part a rule involving a task and a group of others.
The following notations are introduced for a task $i$:

- $St_i$: start time (domain variable) (CHIP++ syntax: I@start)
- $Dur_i$: duration (or processing time) (CHIP++ syntax: I@duration)
- $Est_i$: earliest start time
- $Eft_i$: earliest finish time
- $Lst_i$: latest start time
- $Lft_i$: latest finish time

### 4.2.1. Arc-consistency

**Proposition 1.**

If $Lft_j - Est_j < 2 \times Dur_j + Dur_i$

then $St_i \in [Est_i, Lft_j - Dur_j - Dur_i] \cup [Est_j + Dur_j, Lft_i - Dur_i]$

This rule leads to removing the inconsistent times of $St_i$, i.e., the start times in $[Lft_j - Dur_j - Dur_i, Est_j + Dur_j]$.

**Proof.**

Two sequences are possible between $i$ and $j$: (i before j) or (j before i). If $i$ is before $j$, then $St_i + Dur_i \leq Lft_j - Dur_j$. If $j$ is before $i$, then $St_i \geq Est_j + Dur_j$. It yields:

$St_i \in [Est_i, Lft_j - Dur_j - Dur_i] \cup [Est_j + Dur_j, Lst_i]$. Thus $St_i \notin [Lft_j - Dur_j - Dur_i, Est_j + Dur_j]$ if both previous intervals are disjoint, i.e., if $Lft_j - Dur_j - Dur_i < Est_j + Dur_j \leftrightarrow Lft_j - Est_j < 2 \times Dur_j + Dur_i$. (EOP)

The programming in CHIP of this proposition is very interesting because the same domain variable is used in each hand of the inequality. For example the constraint:

```
J@start #< J@start + I@duration + J@duration
```

examines the maximum value of $St_j$ in the left hand, and the minimum value of $St_j$ in the right hand, yielding:

$Lft_j - Dur_j < Est_j + Dur_i + Dur_j
\leftrightarrow Lst_j - Dur_i < Eft_j$. 
Arc-consistency of this proposition can easily be achieved by using the conditional propagation and the predicate notin/3.

\[
\text{rule}(I,J) :- \\
\quad \text{if } J@\text{start} \leq J@\text{start} + I@\text{duration} + J@\text{duration} \\
\quad \text{then } \text{hole}(I,J), \\
\quad \text{if } I@\text{start} \leq I@\text{start} + J@\text{duration} + I@\text{duration} \\
\quad \text{then } \text{hole}(J,I). \\
\]

\[
\text{hole}(I,J) :- \\
\quad \text{domain_info}(J@\text{start},\text{Est}_j,\text{Lst}_j,_,_,_) \\
\quad \text{Min is } \text{Lst}_j - I@\text{duration} + 1, \\
\quad \text{Max is } \text{Est}_j + J@\text{duration} - 1, \\
\quad \text{notin}(I@\text{start},\text{Min},\text{Max}). \\
\]

**Note.** Value +1 (resp. -1) is added to variable Min (resp. Max) because bound Est_i (resp. Lst_i) still holds for a task i.

**• Precedence constraints**

The following proposition gives conditions over the sequencing between a couple of tasks.

**Proposition 2.**

\[
\text{if } \text{Lft}_j - \text{Est}_i < \text{Dur}_i + \text{Dur}_j \\
\text{then } \text{St}_j < \text{St}_i + \text{Dur}_i
\]

**Proof.**

Suppose we have \( \text{Lft}_j - \text{Est}_i < \text{Dur}_i + \text{Dur}_j \) and by contradiction \( \text{St}_j \geq \text{St}_i + \text{Dur}_i \). The latter inequality is always true if it still holds for \( \text{St}_j = \text{Est}_j \) and \( \text{St}_i = \text{Lst}_i = \text{Lft}_i - \text{Dur}_i \), i.e., \( \text{Est}_j \geq \text{Lft}_i \). Since \( \text{Est}_j \leq \text{Lst}_j = \text{Lft}_j - \text{Dur}_j \) and \( \text{Lft}_i \geq \text{Eft}_i = \text{Est}_i + \text{Dur}_i \), we obtain \( \text{Est}_i + \text{Dur}_i \leq \text{Lft}_j - \text{Dur}_j \) which leads to a contradiction. (EOP)

**Corollary.**

In the disjunctive case where any two tasks cannot overlap:
if $St_j < St_i + Dur_i$
then $St_i \geq St_j + Dur_j$.

Using the conditional propagation mechanism it is very easy to write the condition the start times of tasks sharing the same machine must satisfy.

\[
\text{disjunctive}(I,J) :- \\
\text{if } I@\text{start } < J@\text{start } + J@\text{duration} \\
\text{then } J@\text{start } \geq I@\text{start } + I@\text{duration}, \\
\text{if } J@\text{start } < I@\text{start } + I@\text{duration} \\
\text{then } I@\text{start } \geq J@\text{start } + J@\text{duration}.
\]

### 4.2.2. 1-n–consistency

The previous disjunctive constraint can deduce strong conclusions for the total sequencing. In practice however this rule may be not very active owing to large time windows related to processing times. When pairs of tasks cannot be easily ordered, it may be more interesting to study the extreme positions of a single task relatively to a group of tasks [Esquirol 87][Carlier & P. 88]. More precisely, given the necessity to sequence the whole set of tasks in the case of a disjunctive resource, one can examine whether a task can be processed before or after a given subset of $n$ tasks. For example, if task A can not be placed before both tasks B and C, the only-permitted partial permutations must place A at least after B or C. Subset $\{B,C\}$ is called a **non-posterior set** of tasks relatively to task A. In the same way, we may search for revealing **non-anterior sets** of tasks. Thus, arc-consistency checking for a pair (cf. previous paragraph) appears as a special case of non-anterior or non-posterior sets reduced to a single task. We now write numerical conditions for detecting and propagating such 1-$n$–consistency conditions.

**Proposition 3.**
Given a set $D$ of tasks sharing a disjunctive resource, a task $i \in D$ and a subset $S \subseteq D$ such that $i \notin S$,

\[
\text{if } \max_{j \in S} Lft_j - St_i < Dur_i + \sum_{j \in S} Dur_j \\
\text{then } St_i \geq \min_{j \in S} (St_j + Dur_j)
\]
Proof.
• The “if-part” of the rule states that S is a non-posterior set of tasks relatively to task i. For this reason, an upper bound of the start time over all the permutations of S is calculated: any of them cannot finish after the highest $Lft_j, j \in S$. Since the duration of any permutation of S is at least the sum of the durations of its tasks, any permutation of S cannot start after:

$$\max_{j \in S} Lft_j - \sum_{j \in S} Dur_j.$$  

If $Eft_i$ is greater than this bound, task i cannot be scheduled before set S; S is then proved to be non-posterior to i.

• Since any two tasks of D cannot overlap, the only way to solve this condition is to place i after one task of S:

$$\exists j \in S / St_i \geq St_j + Dur_j.$$  

The “then-part” of the rule states a lower bound of $St_i$ by choosing the lowest earliest finish time among the tasks of S. The proof that this bound is minimal is obvious. Suppose:

$$St_i < \min_{j \in S} (St_j + Dur_j)$$

then $\exists j \in S / St_i \geq St_j + Dur_j$, which contradicts the placement of i after at least one task of S. (EOP)

Notes.
• An analogous proposition symmetrically establishes an upper bound for $St_i$ from a non-anterior set condition.

• At the moment, only maximal subsets S of D are generated in our mock-up ($S = D - \{i\}, \forall i \in D$) in order to avoid a complete but costly naive enumeration of all possible subsets S. Nevertheless, a more clever enumeration exists [Levy et al. 95], which guides the search on the most promising subsets S. That will not be detailed here since it has not been integrated in the mock-up.

Finally, the CHIP implementation of these propositions is also based on a conditional propagation; the $\text{minimum}(X, L)$ and $\text{maximum}(X, L)$ CHIP primitives are used to
calculate the time bounds needed to state non-posterior and non-anterior subsets and then to post new constraints on the start time of each task i of a disjunctive set D.

\[
\text{non_posterior_set}(I,S) :- \\
\quad \text{sum_durations}(S,Sigma), \\
\quad \text{make_ends_list}(S,Ends_List), \\
\quad \text{minimum}(\text{Min\_end},\text{Ends\_List}), \\
\quad \text{maximum}(\text{Max\_end},\text{Ends\_List}), \\
\quad \text{if } \text{Max\_end } \#< I@\text{start } + I@\text{duration } + \text{Sigma} \\
\quad \quad \text{then } I@\text{start } \#\geq \text{Min\_end}.
\]

\[
\text{non_anterior_set}(I,S) :- \\
\quad \text{sum_durations}(S,Sigma), \\
\quad \text{make_starts_list}(S,\text{Starts\_List}), \\
\quad \text{minimum}(\text{Min\_start},\text{Starts\_List}), \\
\quad \text{maximum}(\text{Max\_start},\text{Starts\_List}), \\
\quad \text{if } I@\text{start } \#< \text{Min\_start } + \text{Sigma} \\
\quad \quad \text{then } I@\text{start } + I@\text{duration } \#\leq \text{Max\_start}.
\]

5 RESULTS AND CONCLUDING REMARKS

5.1. Prototyping

Our mock-up interactively builds the sets of parts to be cut out, from the list of production orders [Esquirol et al. 94]. This process is decomposed into two groups of functions:

- parts selection support functions;
- metal sheet filling support functions.

The parts selection support functions

For each part family, the system presents a graphical representation of the order states (relative proportion of each ordered quantity and time staggering curves). From these representations, a subset of parts is extracted, using one or more criteria specified by the
operator (routing, area, delivery date and family). Focusing on this list, the operator can then begin to build the regroupings of parts.

*The metal sheet filling support functions*

The operator defines the parts to be arranged on the same metal sheet. The system verifies their compatibility with the constraints (type, area, geometry, delivery date). If the process fails, the cause is then displayed (insufficient remaining area, incompatible type, outrun delivery date, ...). The metal sheets currently being filled are graphically displayed, as well as their scheduling on the cutting out machine. Updatings are made in real-time. At any time, the operator can go back on all previous decisions, no matter what order they have been taken in.

A hard-copy of a screen proposed by the system is presented below. It shows the lists of processing orders, six metal sheets to realize and a Gantt chart which displays the temporal execution of the aggregated tasks associated with the cutting of parts.

In conclusion, we have seen that using a language such as CHIP may be particularly interesting for prototyping phases in the field of constraint handling. A declarative syntax due to the PROLOG language, high level predicates for developing graphical interfaces and the constraint management mechanism provide to the designers a great flexibility and facility to build mock-ups and prototypes. Moreover, it is very easy to keep up to date an interface in order to integrate new functionalities. Another advantage of CHIP is the facility to design a DSS by using certain predicates giving pieces of information on the variables, the search tree, or the state of the constraints. In previous sections, a few predicates are presented (e.g., domain_info), but we can also cite:

- `touched(Callback, Variable, Term, Type)` : reacts and executes the callback predicate when the modification “Type” is done on the variable “Variable”. It allows the operator to show (e.g., in a graphical way) all modifications on a variable (feedback);

- `pc(Term)` : prints all constraints attached to all variables in term “Term”. In a decision support vision, it may be interesting for example to evaluate the impact of an interaction over a set of variables.
5.2. Conclusions and open issues

The decision support given by the system comes from the combination of, on the one hand, the analysis before decision (by selection and classification of the most appropriated decisions), and on the other hand, the analysis after decision (by inferring the consequences and eliminating wrong values for the remaining variables). The main objective of our approach is to trade on these two kinds of analysis not for an automated resolution but for an incremental resolution controlled by the user. Such an approach seems to be interesting particularly because the number of daily decisions was low in the workshop (despite a high number of constraints). Thus the global resolution time remains short.
Note that instead of the rules presented in Section 4, it would have been possible to use the `cumulative/8` primitive of CHIP. In the disjunctive case however some weaknesses of this primitive have been pointed out. Therefore we decided to implement our own deduction rules. To further improve the propagation process, a new version of our system will integrate rules based on the `energy` concept. The energy of a task is associated with the area a task uses for its execution. It is derived from the product of two dimensions, the time and the resource intensity. The “energy-based approach” is detailed in [Lopez & E. 90], [Lopez 91] and [Lopez et al. 92].

REFERENCES


