CHAPTER 6

CONSTRAINT PROGRAMMING FORMULATION FOR
THE MODELS USING ILOG OPL LANGUAGE

6.1 INTRODUCTION

ILOG Optimization Suite (ILOG 2003, Pinedo 2005), an object-oriented constraint-programming tool marketed by ILOG, has proven to solve resource optimization problems in reasonable computational time and is being used by industries and researchers in a wide range of applications. The ILOG Optimization software provides core engines for applying constraint programming and mathematical programming to a wide variety of problems. The ILOG Optimization Suite has following components: ILOG CPLEX, ILOG Solver, ILOG Scheduler, ILOG Dispatcher, ILOG Configurator, and ILOG OPL Studio. ILOG Solver is one of the core C++ libraries of the ILOG Optimization Suite and provides the basic engine for applying constraint programming to a wide variety of problems. The basic algorithms used for solving any optimization problem in ILOG Solver rely on two simple ideas; the first is to explicitly represent the set of values that a decision variable can take, the second is to represent the search for a solution as a tree reversal. ILOG Scheduler is a C++ library for solving scheduling problems, and supplements ILOG Solver by providing specialized modeling and algorithmic enhancements for problems involving scheduling resources and activities over time. ILOG Scheduler combines constraint programming with advanced scheduling algorithms and heuristics to compute solutions that satisfy both simple and complex finite-capacity scheduling problems. ILOG Scheduler is
widely used for manufacturing scheduling, workforce scheduling, and maintenance scheduling problems. OPL Studio is a complete interactive modeling environment for the rapid development of optimization models and efficient deployment of optimization applications. OPL (Optimization Programming Language), the core of ILOG OPL Studio, is the programming language used to represent problems either in mathematical programming or constraint programming that has access to the libraries of ILOG Solver and ILOG Scheduler.

This Chapter presents constraint programming formulation (CPF) for the two models, FJSP and AJSP, in ILOG OPL language. The coding, illustration and description of the constraint programming for the two models, formulated using ILOG OPL language and solved using ILOG Solver, is explained in the following sections.

6.2 CONSTRAINT PROGRAMMING FORMULATION FOR FJSP

Figure 6.1 shows the CPF for the flexible job shop scheduling problem using the ILOG OPL modeling language. Figure 6.2 shows the coded data for the illustration problem given in Table 4.1 (Chapter 4).

Lines 1 through 17 in the OPL model declare the data for the problem. Line 1 declares the operations of all the jobs as enumerated data \{Oij\}. The notation of “…” in the data declaration indicates that the data of the problem exists in a separate data file. Line 2 declares the total number of machines \(m\). Line 3 uses range, an OPL keyword, to create an identifier \(k\) that ranges from 1 to \(m\). Line 4 declares the processing time of each operation with all the machines. Line 5 declares the set of machines that cannot perform an operation Oij (\(\forall Oij\)). Line 6 through 10 declares two structure members, before and after, each representing an operation Oij. Line 11 through 14
declares a structure member ending that represents an operation Oij. Line 15 declares a set of two operations represented as {<before, after>} to establish precedence relationship between them using the constraint given in lines 40 and 41. Line 16 declares a set of last operation of all the jobs {<ending>} to evaluate makespan time using the constraint given in lines 42 and 43.

```plaintext
enum Operations =...; int m =...; range k 1..m; int* t[Operations, k]=...; [k] cannotperform[Operations]=...; struct precedences { Operations before; Operations after; }; struct precede { Operations ending; }; {precedences} Setofprecedences =...; {precede} Setoflastoper=...; scheduleHorizon=2000; var int duration[Operations] in 0..100; Activity task [O in Operations] (duration[O]); Activity makespan(0); UnaryResource tool[k]; AlternativeResources A(tool); Minimize makespan.end subject to { forall(O in Operations) task[O] requires A; forall(O in Operations) forall(w in cannotperform[O]) not activityHasSelectedResource(task[O],A,tool[w]); forall(O in Operations) forall(w in k) activityHasSelectedResource(task[O],A,tool[w]) => duration[O]=t[O,w]; forall (p in Setofprecedences) task[p.before] precedes task[p.after]; forall (p in Setoflastoper) task[p.ending] precedes makespan; }
```

**Figure 6.1** Constraint programming formulation for FJSP using ILOG OPL modeling language
Operations = \{O_{11},O_{12},O_{13},O_{21},O_{22},O_{23},O_{31},O_{32},O_{33} \};
m=5;
induration=[
    [0,3,7,0,0], [4,0,0,2,0], [1,2,0,0,0],
    [0,5,0,0,2], [0,3,6,0,0], [3,0,0,0,7],
    [0,4,5,0,0], [2,0,0,3,0], [1,0,3,0,0]
];
cannotperform=[ \{1,4,5\}, \{2,3,5\}, \{3,4,5\}, \{1,3,4\}, \{1,4,5\},
    \{2,3,4\}, \{1,4,5\}, \{2,3,5\}, \{2,4,5\} ];
Setofprecedences=\{<O_{11},O_{12}>,<O_{12},O_{13}>,<O_{21},O_{22}>,<O_{22},O_{23}>,<O_{31},O_{32}>,
    <O_{32},O_{33}>\};
Setoflastoper=\{<O_{13}>,<O_{23}>,<O_{33}>\};

**Figure 6.2 Data for the illustration problem**

All scheduling concepts used in ILOG OPL are defined over a global time interval (scheduleOrigin,scheduleHorizon). OPL has default values for both the origin (0) and the horizon (a large number). The horizon in the model is set as 2000 in line 17. Lines 19 through 23 are decision variables for the problem. Line 19 declares an integer variable $duration[O]$ to represent the processing time of each operation on a machine to which it is allocated for processing. The most fundamental concept in OPL for scheduling applications is the *Activity*. An *Activity* is an object containing three data items, a starting date, a duration, and an ending date, together with the duration constraint stating that the ending date is the starting date plus the duration. Line 20 declares $task[O]$ as a set of operations, where each operation in the set is defined as an *Activity* with starting date, ending date and $duration[O]$ as the decision variables. Line 21 declares $makespan$ as an activity with time duration 0. Line 22 declares $tool[k]$ as a set of machines, where each machine in the set is defined as a *UnaryResource* in the model. A unary resource is one that cannot be shared by two activities, i.e., as soon as an activity requires a resource for a time interval, no other activity can use it during the same time interval. Hence this satisfies the constraint that no two activities requiring the same unary resource are scheduled at the same time. Line 23 declares $A(tool)$
as AlternativeResources, an OPL keyword for scheduling applications, represents an array of unary resources. Lines 25 and 26 indicate the objective of the problem. The objective is to minimize the completion of the activity makespan. Lines 28 through 43 describe the constraints of the problem. Lines 30 and 31 uses the keyword requires to specify that each activity in task[O] can be performed by any of the unary resources in A. Lines 32 through 34 uses not activityHasSelectedResource, an OPL keyword, to specify that each activity in task[O] does not select a resource w in cannotperform among the alternate resources A. Lines 35 through 37 imposes that activity task[O] selects a resource w in k among the alternative resources A and assigns the processing time to duration[O]. OPL has a specific keyword precedes for expressing precedence constraint between two activities. Lines 39 and 40 uses the keyword precedes to impose the precedence relationship between two activities declared as Setofprecedesences in line 15. Lines 41 and 42 imposes that the set of last operation of all jobs {OiJi} represented as tool[p.ending] precedes the activity makespan. The output obtained with ILOG Solver for the illustration problem is shown in Figure 6.3 and 6.4.
Optimal Solution with Objective Value: 10

duration[O11] = 3
duration[O12] = 2
duration[O13] = 1
duration[O21] = 2
duration[O22] = 3
duration[O23] = 3
duration[O31] = 5
duration[O32] = 3
duration[O33] = 1
task[O11] = [0 -- 3 --> 3]
task[O12] = [3 -- 2 --> 5]
task[O13] = [5 -- 1 --> 6]
task[O21] = [0 -- 2 --> 2]
task[O22] = [3 -- 3 --> 6]
task[O23] = [6 -- 3 --> 9]
task[O31] = [0 -- 5 --> 5]
task[O32] = [5 -- 3 --> 8]
task[O33] = [9 -- 1 --> 10]
makespan = [10 -- 0 --> 10]
tool[1] = Unary Resource
  required by task[O13] over [5,6] in capacity 1
  required by task[O23] over [6,9] in capacity 1
  required by task[O33] over [9,10] in capacity 1

tool[2] = Unary Resource
  required by task[O11] over [0,3] in capacity 1
  required by task[O22] over [3,6] in capacity 1

tool[3] = Unary Resource
  required by task[O31] over [0,5] in capacity 1

tool[4] = Unary Resource
  required by task[O12] over [3,5] in capacity 1
  required by task[O32] over [5,8] in capacity 1

tool[5] = Unary Resource
  required by task[O21] over [0,2] in capacity 1

A = AlternativeResources

Figure 6.3  Output obtained with ILOG OPL Studio for the illustration problem
Figure 6.4 Gantt chart generated by ILOG Solver for the illustration problem
6.3 CONSTRAINT PROGRAMMING FORMULATION FOR AJSP

Figure 6.5 shows the CPF for the assembly job shop scheduling problem associated with multiple routings using the ILOG OPL modeling language. Figure 6.6 shows the coded data for the illustration problem given in Tables 5.1 and 5.2 (Chapter 5).

Lines 1 through 26 in the OPL model declare the data for the problem. Line 1 declares the processing operations of all the components as enumerated data \{Opij\}. Line 2 declares the assembly operations of all the products as enumerated data \{Op0j\}. Line 3 declares each product as enumerated data \{Pp0\}. Line 4 declares the total number of machines \(m\). Line 5 uses the OPL keyword range to create an identifier \(k\) that ranges from 1 to \(m\). Line 6 declares the processing time of each processing operation with all the machines. Line 7 declares the assembling time of each assembly operation. Line 8 declares the due date for each product. Line 9 declares the penalty cost per unit time of tardiness for each product. Line 10 declares the set of machines that cannot perform operation Opij (\(\forall Opij\)). Lines 11 through 13 declare two structure members, before1 and after1, each representing a processing operation Opij.
Figure 6.5 Constraint programming formulation for AJSP using ILOG OPL modeling language
Operations = \{O_{111}, O_{112}, O_{113}, O_{121}, O_{131}, O_{132}, O_{211}, O_{221}, O_{231}, O_{232}, O_{233}, O_{241}, O_{242}, O_{251}, O_{252}\};

Assembly = \{O_{101}, O_{201}, O_{202}, O_{203}\};

Products = \{P_{10}, P_{20}\};

m=5;

induration = [\begin{bmatrix}
2.5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 4 & 0 \\
0 & 0 & 3 & 0 & 2 \\
0 & 2 & 0 & 3 & 0 \\
5 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & 3 \\
0 & 4 & 3 & 0 & 2 \\
0 & 0 & 0 & 3 & 0 \\
4 & 0 & 5 & 0 & 0 \\
0 & 0 & 0 & 0 & 7 \\
0 & 3 & 4 & 0 & 0 \\
0 & 0 & 0 & 3 & 0 \\
0 & 0 & 0 & 0 & 2 \\
0 & 0 & 3 & 0 & 4 \\
3 & 0 & 2 & 0 & 0
\end{bmatrix}];

aduration = [5, 2, 4, 3];

duedate = [10, 20];

tardicost = [1, 2];

cannotperform = [\begin{bmatrix}
3 & 4 & 5 & 1 & 2 & 3 & 4 \\
1 & 2 & 4 & 1 & 3 & 5 & 1 \\
2 & 3 & 4 & 1 & 4 & 1 & 2 & 3 & 5 \\
2 & 4 & 5 & 1 & 2 & 3 & 5 & 1 & 2 & 3 & 4 \\
1 & 2 & 4 & 1 & 4 & 1 & 2 & 4 & 5
\end{bmatrix}];

Setoflastoper = \{<O_{101}>, <O_{203}>\};

Setofprecedence1 = \{<O_{111}, O_{112}>, <O_{112}, O_{113}>, <O_{131}, O_{132}>, <O_{231}, O_{232}>, <O_{232}, O_{233}>, <O_{241}, O_{242}>, <O_{251}, O_{252}>\};

Setofprecedence2 = \{<O_{113}, O_{101}>, <O_{121}, O_{101}>, <O_{132}, O_{101}>, <O_{211}, O_{201}>, <O_{221}, O_{201}>, <O_{242}, O_{202}>, <O_{252}, O_{202}>, <O_{233}, O_{203}>\};

Setofprecedence3 = \{<O_{201}, O_{203}>, <O_{202}, O_{203}>\};

\textbf{Figure 6.6 Data for the illustration problem}
Lines 14 through 16 declare two structure members, before2 and after2, where before2 represents a processing operation Opij and after2 represents an assembly operation Op0j. Lines 17 through 19 declare two structure members, before3 and after3, each representing an assembly operation Op0j. Lines 20 and 21 declare a structure member ending that represents an assembly operation Op0j. Line 22 declares a set of two processing operations represented as \( \{<\text{before1}, \text{after1}>\} \) to establish precedence relationship between them using the constraint given in lines 49 and 50. Line 23 declares a set of two operations (a processing operation and an assembly operation) represented as \( \{<\text{before2}, \text{after2}>\} \), to establish precedence relationship between them using the constraint given in lines 51 and 52. Line 24 declares a set of two assembly operations represented as \( \{<\text{before3}, \text{after3}>\} \) to establish precedence relationship between them using the constraint given in lines 53 and 54. Line 25 declares a set of final assembly operations of all the products \( \{<\text{ending}>\} \) to determine the completion time of each product using the constraint given in lines 55 through 57. The horizon in the model is set as 2000 in line 26. Lines 28 through 34 are decision variables for the problem. Line 28 declares an integer variable \( \text{duration}[O] \) to represent the processing time of each operation on a machine to which it is allocated for processing. Line 29 declares an integer variable \( \text{tardiness}[P] \) to represent the tardiness of each product. Line 30 declares \( \text{task}[O] \) as a set of processing operations, where each operation in the set is defined as an Activity with starting date, ending date and \( \text{duration}[O] \) as the decision variables. Line 31 declares \( \text{Assem}[O] \) as a set of assembly operations, where each operation in the set is defined as an Activity. Line 32 declares \( \text{tool}[k] \) as a set of machines, where each machine in the set is defined as a UnaryResource in the model. Line 33 declares \( \text{A(tool)} \) as AlternativeResources, an OPL keyword for scheduling applications, represents an array of unary resources. Lines 35 and 36 indicate the objective of the problem. The objective is to minimize the sum of tardiness cost of all
the products. Lines 38 through 60 describe the constraints of the problem. Lines 40 and 41 use the keyword `requires` to specify that each activity in `task[O]` can be performed by any of the unary resources in `A`. Lines 42 through 44 use the OPL keyword `not activityHasSelectedResource` to specify that each activity in `task[O]` does not select a resource `w` in `cannotperform` among the alternate resources `A`. Lines 45 through 48 impose that activity `task[O]` selects a resource `w` in `k` among the alternative resources `A` and assigns the processing time to `duration[O]`. Lines 49 through 54 use the keyword `precedes` to impose the precedence relationship between two activities declared in `Setofprecedence1`, `Setofprecedence2` and `Setofprecedence3`. Lines 55 through 57 impose that tardiness of each product `tardiness[P]` is equal to the difference between the completion time of its final assembly operation `Assem{s.ending}.end` and the due date `duedate[P]`. Lines 60 through 66 describe a search strategy for the problem. The search space is explored by trying to assign each processing `Activity` to the resource with lower duration first (lines 62 through 65) while looking for a solution. The output obtained with ILOG Solver for the illustration problem is shown in Figure 6.7 and Figure 6.8.
<table>
<thead>
<tr>
<th>Optimal Solution with Objective Value: 7</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>duration[O111] = 2</td>
<td>task[O242] = [4 -- 2 --&gt; 6]</td>
</tr>
<tr>
<td>duration[O112] = 4</td>
<td>task[O251] = [0 -- 4 --&gt; 4]</td>
</tr>
<tr>
<td>duration[O113] = 3</td>
<td>task[O252] = [7 -- 3 --&gt; 10]</td>
</tr>
<tr>
<td>duration[O121] = 2</td>
<td>Assem[O101] = [12 -- 5 --&gt; 17]</td>
</tr>
<tr>
<td>duration[O131] = 5</td>
<td>Assem[O201] = [15 -- 2 --&gt; 17]</td>
</tr>
<tr>
<td>duration[O132] = 2</td>
<td>Assem[O202] = [10 -- 4 --&gt; 14]</td>
</tr>
<tr>
<td>duration[O211] = 2</td>
<td>Assem[O203] = [17 -- 3 --&gt; 20]</td>
</tr>
<tr>
<td>duration[O221] = 3</td>
<td>tool[1] = Unary Resource</td>
</tr>
<tr>
<td>duration[O231] = 5</td>
<td>required by task[O111] over [0,2] in capacity 1</td>
</tr>
<tr>
<td>duration[O233] = 3</td>
<td>required by task[O252] over [7,10] in capacity 1</td>
</tr>
<tr>
<td>duration[O242] = 2</td>
<td>required by task[O121] over [0,2] in capacity 1</td>
</tr>
<tr>
<td>duration[O251] = 4</td>
<td>required by task[O233] over [13,16] in capacity 1</td>
</tr>
<tr>
<td>tardiness[P10] = 7</td>
<td>required by task[O113] over [7,10] in capacity 1</td>
</tr>
<tr>
<td>tardiness[P20] = 0</td>
<td>required by task[O132] over [10,12] in capacity 1</td>
</tr>
<tr>
<td>task[O111] = [0 -- 2 --&gt; 2]</td>
<td>required by task[O231] over [0,5] in capacity 1</td>
</tr>
<tr>
<td>task[O113] = [7 -- 3 --&gt; 10]</td>
<td>required by task[O112] over [3,7] in capacity 1</td>
</tr>
<tr>
<td>task[O121] = [0 -- 2 --&gt; 2]</td>
<td>required by task[O221] over [7,10] in capacity 1</td>
</tr>
<tr>
<td>task[O131] = [2 -- 5 --&gt; 7]</td>
<td>required by task[O241] over [0,3] in capacity 1</td>
</tr>
<tr>
<td>task[O221] = [7 -- 3 --&gt; 10]</td>
<td>required by task[O232] over [6,13] in capacity 1</td>
</tr>
<tr>
<td>task[O231] = [0 -- 5 --&gt; 5]</td>
<td>required by task[O242] over [4,6] in capacity 1</td>
</tr>
<tr>
<td>task[O232] = [6 -- 7 --&gt; 13]</td>
<td>required by task[O251] over [0,4] in capacity 1</td>
</tr>
<tr>
<td>task[O233] = [13 -- 3 --&gt; 16]</td>
<td>A = AlternativeResources</td>
</tr>
<tr>
<td>task[O241] = [0 -- 3 --&gt; 3]</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.7 Output obtained with ILOG Solver for the illustration problem
Figure 6.8 Activity chart generated by ILOG Solver for the illustration problem