ILOG OPL Studio

Technical overview

Sofiane Oussedik
soussedik@ilog.fr
Outline

ILOG OPL Studio

• Overview
• Models and Data
• Linear Programming Example
• Constraint Programming Example
• Scheduling Example
• Data Management Features
• Iterative Applications
• OPL Component Libraries
ILOG OPL Studio reduces time to market for developing optimization applications
A Common API for CPLEX and Solver
Inside OPL Studio

ILOG OPL Studio

• **OPL Optimization Programming Language**
  - A language for representing optimization problems
  - Has advanced types to allow better organization of data
  - Supports constraint programming, linear and integer programming, and scheduling problems
  - Database and Microsoft Excel connectivity
  - OPLScript for iterative solving and hybrid optimization

• **Graphical User Interface for Optimization Problems**
  - Text editor with keyword colors for entering problems and data
  - Visualizations of data and solutions
  - Menus/buttons for controlling optimization
  - Online help for OPL language
OPL Component Libraries

ILOG OPL Studio

- API's for embedding OPL models and OPL scripts
  - C++
  - Microsoft COM / .NET
    - Visual Basic
    - Visual Basic for Applications (Excel, Access, etc.)
  - Java
  - Web
    - ASP, JSP
- Links with ILOG CPLEX, Solver, and Scheduler
OPL features for development
What Are Models?

Linear programming

- A data-independent abstraction of a problem
- OPL lets you write down the mathematical representation of a model separately from the data
Linear programming

- A manufacturer wants to sell a product
- The product can be made either
  - Inside the factory
    - Scarce resources are required to build the product
    - There is a cost per unit to manufacture the product
  - Outside the factory
    - There is a cost per unit to purchase the product
- All demand must be satisfied
- The goal is to minimize cost
Data Declarations

Linear programming

- Sets of products and resources
  ```
  enum Products ...;
  enum Resources ...;
  ```
- Number of units of each resource needed to produce one unit of each product
  ```
  float+ consumption[Products, Resources] = ...;
  ```
- Total number of available resources
  ```
  float+ capacity[Resources] = ...;
  ```
- Number of units in demand for each product
  ```
  float+ demand[Products] = ...;
  ```
- Cost per unit of inside and outside production
  ```
  float+ insideCost[Products] = ...;
  float+ outsideCost[Products] = ...;
  ```
Products Could Be Jewelry

Linear programming

- **Products and Resources**
  
  \[
  \text{Products} = \{ \text{rings earrings} \}; \\
  \text{Resources} = \{ \text{gold diamonds} \};
  \]

- **Consumption**
  
  - A ring requires 3 units of gold and 1 diamond
  - A set of earrings requires 2 units of gold and 2 diamonds
  
  \[
  \text{consumption} = \left[ \begin{bmatrix} 3, 1 \end{bmatrix}, \begin{bmatrix} 2, 2 \end{bmatrix} \right];
  \]

- **Capacity (Available units of gold and diamonds)**
  
  \[
  \text{capacity} = \left[ \begin{bmatrix} 130 \end{bmatrix}, \begin{bmatrix} 180 \end{bmatrix} \right];
  \]

- **Demand (Number of rings and earrings)**
  
  \[
  \text{demand} = \left[ \begin{bmatrix} 100 \end{bmatrix}, \begin{bmatrix} 150 \end{bmatrix} \right];
  \]

- **Costs (per unit for rings and earrings)**
  
  \[
  \text{insideCost} = \left[ \begin{bmatrix} 250 \end{bmatrix}, \begin{bmatrix} 200 \end{bmatrix} \right]; \\
  \text{outsideCost} = \left[ \begin{bmatrix} 260 \end{bmatrix}, \begin{bmatrix} 270 \end{bmatrix} \right];
  \]
### Linear programming

- **Products and Resources**
  
  \[
  \text{Products} = \{ \text{kluski, capellini, fettucine} \}; \\
  \text{Resources} = \{ \text{flour, eggs} \};
  \]

- **Consumption**
  
  - Kluski requires 0.5 units of flour and 0.2 eggs
  - Capellini requires 0.4 units of flour and 0.4 eggs
  - Fettucine requires 0.3 units of flour and 0.6 eggs
  
  \[
  \text{consumption} = [ [0.5, 0.2], [0.4, 0.4], [0.3, 0.6] ];
  \]

- **Capacity (Available units of flour and eggs)**
  
  \[
  \text{capacity} = [20, 40];
  \]

- **Demand (Number of each pasta needed)**
  
  \[
  \text{demand} = [100, 200, 300];
  \]

- **Costs (per unit for each pasta)**
  
  \[
  \text{insideCost} = [0.6, 0.8, 0.3]; \\
  \text{outsideCost} = [0.8, 0.9, 0.4];
  \]
Problem Model Is Identical

Linear programming

```plaintext
data
enum Products ...;
enum Resources ...;

float+ consumption[Products, Resources] = ...;
float+ capacity[Resources] = ...;
float+ demand[Products] = ...;
float+ insideCost[Products] = ...;
float+ outsideCost[Products] = ...;

variables
var float+ inside[Products];
var float+ outside[Products];
```
Problem Model Is Identical (2)

Linear programming

minimize
\[ \sum_{p \in \text{Products}} (\text{insideCost}[p] \times \text{inside}[p] + \text{outsideCost}[p] \times \text{outside}[p]) \]

subject to {
    forall(r in Resources)
    \[ \sum_{p \in \text{Products}} \text{consumption}[p, r] \times \text{inside}[p] \leq \text{capacity}[r] \];

    forall(p in Products)
    \[ \text{inside}[p] + \text{outside}[p] \geq \text{demand}[p] \];
}
Demo of production.prj

Linear programming

- Run production.prj
- Display inside and outside production
Linear and Integer Programming

Linear programming

- OPL can represent linear and mixed integer programming problems
  - Objective functions and constraints are linear
    - E.g., $5x + 3y - 4z$
  - Variables can be floating point or integer valued
  - Constraints are inequalities or equalities ($\leq, \geq, =$)
- OPL Studio can solve
  - Linear Programs with any of the CPLEX algorithms
  - Mixed Integer Programs using the CPLEX MIP algorithm
  - Mixed Integer Programs by using constraint programming-based search
- OPL provides access to all CPLEX algorithmic settings
Map Coloring Example

Constraint programming

• Given a list of countries

```
enum Country {Belgium, Denmark, France, Germany, Netherlands, Luxembourg};
```

• A set of colors to assign to countries on the map

```
enum Colors {blue, red, yellow, gray};
```

• Want to decide how to assign the colors to the countries so that no two bordering countries have the same color

```
var Colors color[Country];
```

The decision variables are values from a set
```csharp
enum Country {Belgium, Denmark, France, Germany, Netherlands, Luxembourg};
enum Colors {blue, red, yellow, gray};

var Colors color[Country];

solve {
    color[France] <> color[Belgium];
    color[France] <> color[Luxembourg];
    color[France] <> color[Germany];
    color[Luxembourg] <> color[Germany];
    color[Luxembourg] <> color[Belgium];
    color[Belgium] <> color[Netherlands];
    color[Belgium] <> color[Germany];
    color[Germany] <> color[Netherlands];
    color[Germany] <> color[Denmark];
}
```
Demo of map.mod

Constraint programming

- Run map.mod
- Show all solutions
- Rerun to show the propagation
  - Select Execution > Browse Active Model
  - Right-click on Variables > color and select display domain
  - Select Debug > Stop at Choice Point
  - Step through the model
  - Disable Debug > Stop at Choice Point when finished
Visualizing Solutions

Constraint programming

• Develop application-specific graphical output using the drawing board
  • Lines
  • Polygons
  • Arcs, Circles
  • Text labels

• Graphics can be added to the search procedure
  • Draw an object when variables are fixed to values
  • Graphics are updated automatically during the search procedure
Demo of mapgr.prj

- Run mapgr.prj
- Show all solutions
- Graphics help illustrate how many neighbors constrain each country
OPL allows you to represent problems using constraint programming features

- Variables can be set to a value from a set
- A variable can index into an array (\( y[x[k]] \))
- Constraint relations can be strict inequalities (<, >, < >)
- Logical conditions can be modeled
  \[ x < 4 \Rightarrow y > 5 \]
- Global constraints can be written
  \texttt{alldifferent}(x)
- Constraints can have values (meta constraints)
  \[ \text{sum} (j \text{ in } \text{myset}) (x[j] > 5) = 3 \]
OPL Studio uses ILOG Solver to solve constraint programming problems
  - Default search strategy need not be programmed
  - Users can program their own search strategies

OPL Studio allows debugging of search strategies
  - Users can visualize the values of their variables
  - Users can step through a search procedure
  - The search tree can be visualized
• Want to assign S stores to W warehouses. The problem is as follows:
  • The cost of assigning store \( s \) to warehouse \( w \) is given by the array element \( \text{supplyCost}[s,w] \).
  • Each warehouse \( w \) can have at most \( \text{capacity}[w] \) stores assigned to it.
  • There is a fixed cost \( \text{fixed}=30 \) for opening up each warehouse.
var int open[Warehouses] in 0..1;
var int supply[Stores,Warehouses] in 0..1;

minimize
    sum(w in Warehouses) fixed * open[w] +
    sum(w in Warehouses, s in Stores)
        supplyCost[s,w] * supply[s,w]
subject to {
    forall(s in Stores)
        sum(w in Warehouses) supply[s,w] = 1;
    forall(w in Warehouses, s in Stores)
        supply[s,w] <= open[w];
    forall(w in Warehouses)
        sum(s in Stores) supply[s,w] <= capacity[w];
};
var int open[Warehouses] in 0..1;
var Warehouses supplier[Stores];
var int cost[Stores] in 0..maxCost;

minimize
  sum(s in Stores) cost[s] +
  sum(w in Warehouses) fixed * open[w]
subject to {
  forall(s in Stores)
    cost[s] = supplyCost[s,supplier[s]];
  forall(s in Stores )
    open[supplier[s]] = 1;
  forall(w in Warehouses)
    sum(s in Stores) (supplier[s] = w) <= capacity[w];
};

search {
  forall(s in Stores ordered by decreasing regret[min(cost[s])])
    tryall(w in Warehouses ordered by increasing supplyCost[s,w])
      supplier[s] = w;
};
MIP versus CP formulation

• **Decision variables**
  - The constraint programming formulation has $2S+W$ decision variables.
  - The mixed integer formulation has $SW+W$ decision variables.
  - The CP formulation has a decision variable over a finite set of values to represent the cost of shipping for store $s$.
  - The MIP formulation represents the cost of shipping for store $s$ as an implied expression.
    
    $\sum (w \text{ in } \text{Warehouses}) \text{ supplyCost}[s,w] \ast \text{supply}[s,w]$

• **Expressions**
  - The CP formulation uses expressions of the form $\text{open}[\text{supplier}[s]]$, which uses a decision variable to index into another decision variable.
  - The CP formulation uses the expression $(\text{supplier}[s] = w)$ that evaluates to a 0/1 value.
search {
    forall(s in Stores ordered by decreasing regretmin(cost[s]))
    tryall(w in Warehouses ordered by increasing supplyCost[s,w])
    supplier[s] = w;
};

• *cost*[s] can only take on values from *supplyCost*[s,w] for the set of open warehouses w
  
  \[ \text{regretmin} = (\text{second lowest value}) - (\text{lowest value}) \]

• Pick the store with the largest regret, then pick the warehouse with the smallest cost
• Then open that warehouse
But Which is BETTER??

• It depends upon the data
• It depends on the search strategy
• It depends on the combinatorial nature of the problem

• For general applications, you need tools that allow you to try both methodologies!
Scheduling Example (Part 1)

Scheduling

• Have to schedule a set of tasks to build a house

```c
enum Tasks { masonry, carpentry, plumbing,
            ceiling, roofing, painting,
            windows, facade, garden, moving };
```

• Tasks require a given amount of time to be completed

```c
int duration[Tasks] = [7,3,8,3,1,2,1,2,1,1];
int totalDuration =
    sum(t in Tasks) duration[t];
scheduleHorizon = totalDuration;
```

• Some tasks require other tasks to be completed

```c
task[masonry] precedes task[ceiling];
task[carpentry] precedes task[roofing];
task[ceiling] precedes task[painting];
...
Scheduling Example (Part 2)

Scheduling

• A group of workers must build the house, but each cannot perform certain tasks

```cpp
def enum Workers { joe, jack, jim };
{Workers} cannotperform[Tasks] = #[
  masonry: { jim },
  carpentry: { jack },
  plumbing: { joe, jim },
  ceiling: { jack },
  roofing: { jack },
  painting: { joe },
  windows: { jack },
  facade: { jim },
  garden: {},
  moving: { jack } ]#;
```

• Goal is to schedule the tasks and the workers and to minimize the maximum amount of time any worker is on duty
enum Tasks { masonry, carpentry, plumbing, 
  ceiling, roofing, painting, 
  windows, facade, garden, moving }; 
int duration[Tasks] = [7,3,8,3,1,2,1,2,1,1]; 
int totalDuration = sum(t in Tasks) duration[t]; 
scheduleHorizon = totalDuration; 
enum Workers { joe, jack, jim }; 
{Workers} cannotperform[Tasks] = #[
  masonry: { jim }, carpentry: { jack },
  plumbing: { joe, jim }, ceiling: { jack },
  roofing: { jack }, painting: { joe },
  windows: { jack }, facade: { jim },
  garden: {}, moving: { jack } ]#;
Activity task[t in Tasks](duration[t]); 
var int durationWorkers[Workers] in 0..totalDuration; 
Activity attendance[w in Workers](durationWorkers[w]); 
UnaryResource worker[Workers]; 
AlternativeResources s(worker);
Objective Function

Scheduling

minimize max(w in Workers) durationWorkers[w]

subject to {
    task[masonry] precedes task[carpentry];
    task[masonry] precedes task[plumbing];
    task[ceiling] precedes task[paint];
    task[roof] precedes task[plumbing];
    task[roof] precedes task[facade];
    task[roof] precedes task[garden];
    task[roof] precedes task[windows];
    task[roof] precedes task[facade];
    task[roof] precedes task[moving];
    forall(t in Tasks) task[t] requires s;
    forall(t in Tasks)forall(w in cannotperform[t])
        not activityHasSelectedResource(task[t],s,worker[w]);
    forall(t in Tasks)forall(w in Workers)
        activityHasSelectedResource(task[t],s,worker[w]) =>
            attendance[w].start <= task[t].start &
            attendance[w].end >= task[t].end;
};

Constraints
Demo of house3.mod

Scheduling

• Run house3.mod

• Show the Gantt chart for task activities and worker resources
Scheduling Models

Scheduling

- OPL supports modeling entities for scheduling problems
  - Activities
    - Have durations
    - Can be breakable
    - Constraints to state that activities precede other activities
  - Resources
    - Unary Resources
    - Discrete Resources
    - Reservoirs
    - State Resources with Transition Times
- Can also add other kinds of constraints using constraint programming
Structured Data

Data Management

- **Given a set of cities**
  ```
  enum Cities { MIA, EWR, SFO, BOS };
  ```

- **Origin/destination pairs can be represented as**
  ```
  struct Pair { Cities o; Cities d; };
  ```

- **A set of pairs would be represented as**
  ```
  setof(Pair) odpairs =
  {<MIA,EWR>,<MIA,SFO>,<SFO,BOS>,<EWR,SFO>};
  ```

- **The origins can be computed as**
  ```
  setof(Cities) origins = { o | <o,d> in odpairs};
  ```

- **A decision variable array over the set of odpairs would be written as:**
  ```
  var float+ flow[odpairs];
  ```
Database connectivity via SQL

Data Management

• Reading

```c
struct Precedence {
    string before;  //Task
    string after;   //Task
};
setof(Precedence) precedences from
    DBRead(db, "select * from PRECEDENCE");
```

• Writing

```c
struct Schedule {
    string task;
    int startTime;
    int endTime;
};
setof(Schedule) resultSet = { /* Set of results */ }; DBUpdate(db, "insert into Result
    (task, startTime, endTime)
    values (?, ?, ?)") (resultSet);
```
Excel connectivity

Data Management

- **Reading**

  setof(int) TimePeriods from SheetRead(sheetData,"Time");
  float+ avail[TimePeriods] from SheetRead(sheetData,"avail");
  setof(string) Products from SheetRead(sheetData,"Product");
  float+ revenue[Products,TimePeriods] from SheetRead(sheetData,"Revenue");

<table>
<thead>
<tr>
<th>Time</th>
<th>avail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
</tr>
</tbody>
</table>

- **Writing**

  var float+ Make[Products,TimePeriods];
  SheetWrite(sheetResult,"A2:D3") (Make);

- **Use named ranges or sheet references**

<table>
<thead>
<tr>
<th>Revenue</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>bands</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>coils</td>
<td>30</td>
<td>35</td>
<td>37</td>
<td>39</td>
</tr>
</tbody>
</table>
OPLScript

Iterative Applications

• Scripting language to control execution of OPL models and solution methods

• Applications:
  • Solve a sequence of related models
  • Solve a model with varying data
  • Complex decomposition strategies
  • Hybrid optimization
OPLScript Example

Iterative Applications

```c++
Model produce("mulprod.mod","mulprod.dat") editMode;
import enum Resources produce.Resources;
int+ capFlour := produce.capacity[flour];

forall(i in 1..4) {
    produce.capacity[flour] := capFlour;
    produce.solve();
    cout << "Flour capacity" << capFlour "Objective Function: " <<
         produce.objectiveValue() << endl;
    Basis b(produce);
    produce.reset();
    produce.setBasis(b);
    capFlour := capFlour + 1;
}
```

Declare Model

Set data, then Solve!

Output Answer

Try additional capacity, using advanced basis
OPL features for deployment
Deployment Options

OPL Component Libraries

---

**Compile**

- OPL model
- Generate compiled model
- Read compiled model
  - VB, Java, C++, Web
- Application with Fixed Model

**Interpret**

- Application reads interpreted scripts and models from files
- Read script
  - VB, Java, C++, Web
- OPL Script
- OPL model
Excel Demo

Steel Production Spreadsheet Application

- Open steelt2D.xls
- Show optimization
- Show how if numbers change, answer changes
**Visual Basic Example**

**OPL Component Libraries**

- **Initialize solver**
  
  Dim solver As COPLsolver
  Set solver = New COPLsolver

- **Model and data loaded from file or memory**
  
  Call solver.loadCompiledModelFileAndDataFile("mulprod.opl", "mulprod.dat", 1)

- **Modeling entities are accessed via strings**
  
  Dim capacity As IOPLarray
  Set capacity = solver.getArray("capacity")
  Dim resources As IOPLenum
  Set resources = solver.getEnum("Resources")
  Dim flour As IOPLenumValue
  Set flour = resources.getValue("flour")
  Dim capFlour As IOPLint
  Set capFlour = capacity.getInt(flour)
C++ and Java almost identical

OPL Component Libraries

- **Initialize solver**
  - **C++**
    ```cpp
    OPLSolver solver;
    ```
  - **Java**
    ```java
    OPLSolver solver = new OPLSolver();
    ```

- **Load model and data from file or memory**
  ```java
  solver.loadCompiledModelFileAndDataFile("mulprod.opl", "mulprod.dat", 1);
  ```

- **Modeling entities are accessed via strings**
  ```java
  OPLArray capacity = solver.getArray("capacity");
  OPLEnum resources = solver.getEnum("Resources");
  OPLEnumValue flour = resources.getValue("flour");
  OPLLInt capFlour = capacity.getInt(flour);
  ```
Web JSP Demo

Steel Production Spreadsheet Application

- Open Staffing Demo
- Show optimization
OPL Studio Key Features

Summary

• Powerful Modeling Language (OPL) for
  • Constraint Programming
  • Scheduling
  • Linear and Mixed Integer Programming

• Graphical User Interface for
  • Entering Models and Data
  • Organizing Projects
  • Visualizing data and solutions
  • Controlling optimization

• Linked with ILOG optimization tools
  • ILOG Solver, Scheduler and CPLEX

• Database access, Spreadsheet access and Scripting
• OPL Component Libraries accelerate application deployment
Rapid Application Deployment

Summary

• Develop your model in OPL Studio, maintaining model/data separation

• Refine your algorithm with OPL search strategies
  • Use visualization to enhance understanding
  • Step through search procedures
  • Use iterative or hybrid approaches

• Incorporate model via OPL Component Libraries
  • Use C++, Visual Basic, Java, ASP, JSP
  • Integrate external data
  • Use the answer wherever needed

• Links with ILOG CPLEX, Solver, Scheduler
Web seminar Series: soon on webseminar.ilog.com

- Overview of CPLEX
- Building CPLEX applications using OPL Studio
- Getting started with OPL Studio

Email: seminarinfo@ilog.com

Visit: http://www.ilog.com