Discovering Combinatorial Optimization with the ILOG Optimization Suite

Gregory Glockner, Ph.D.
Optimization Product Manager
ILOG, Inc.
gglockner@ilog.com
Overview

Agenda

- About ILOG
- Brief introduction to constraint programming
- Exploring combinatorial optimization
- Exploring iterative optimization
- Discrete scheduling
About ILOG

World Leader in Software Components

- Founded 1987
- 590 employees
- 2,000+ customers
- Selling in 30 countries
- NASDAQ/Euronext

ISV/OEM Partners

<table>
<thead>
<tr>
<th>Year</th>
<th>70</th>
<th>120</th>
<th>200</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
About ILOG

ILOG Core Technologies

Visualization

Business Rules

Optimization

Views Component Suite
JViews Component Suite
JTGO
Rules
JRules
Solver
CPLEX
ILOG Optimization Suite

- ILOG OPL Studio
- ILOG Scheduler
- ILOG Dispatcher
- ILOG Configurator
- ILOG JConfigurator

- ILOG CPLEX
  - Hybrid: ILOG Solver & ILOG JSolver
  - ILOG Concert Technology (C++ & Java)
About ILOG

ILOG Optimization Suite – 2

- **Core Engines**
  - ILOG Solver - Constraint Programming Engine
  - ILOG CPLEX - Math Programming Engine

- **Vertical Engine Extensions**
  - ILOG Scheduler - Constraint-Based Scheduling
  - ILOG Dispatcher - Vehicle Routing, Technician Dispatching
  - ILOG Configurator - Product and Service Configuration

- **Modeling Tools**
  - OPL Studio - Rapid Development of Optimization Apps
  - AMPL - Modeling Support for CPLEX

We use OPL Studio here since its high level language makes it an easy starting point
Inside ILOG OPL Studio

- OPL: Optimization Programming Language
  - High-level language specialized for optimization modeling
- OPLScript
  - Procedural language to control optimization process
- OPL Studio
  - IDE to edit and solve models and to view results
- OPL Component Libraries
  - APIs to integrate models into standalone applications via C++, Java, and Microsoft COM and .NET (C#, VB, VBA)
- Direct links to the ILOG Optimization Suite
  - ILOG CPLEX, ILOG Solver, ILOG Scheduler
Only ILOG has optimization technology for the entire planning horizon.
Overview

Agenda

- About ILOG
- Brief introduction to constraint programming
- Exploring combinatorial optimization
- Exploring iterative optimization
- Discrete scheduling
Find a solution to:
\[ x + y < 5, \ x \neq y, \ x, y \in \{1, \ldots, 5\} \]
CP in a Nutshell

- **Essential problem:** To satisfy a given set of constraints by assigning values to variables
  - Typical problems involve discrete variables
- **Enumerate values via branch and bound**
  - Try all values in domain
  - Use general and custom searches
- **Constraint propagation reduces domain at each node**
  - Eliminate inconsistent values from current partial solution
Comparing CP and IP

- **Computation at nodes**
  - CP: Constraint propagation is fast but naïve
  - IP: LP relaxation is slow, powerful algorithm

- **Constraint representation**
  - IP uses linear inequalities
  - CP uses logical constraints

- **Search methods**
  - IP has built-in branch-and-bound algorithm
  - User supplies CP search
Some CP Constraints – 1

- Logical and high-order
  - \((x > 0) \Rightarrow (y = 0)\)
  - \((Q = 0) \lor (Q \geq 100)\)
  - \(C = 5 \times Q + 1000 \times (Q > 0)\)

- Element and table
  - \(T = \sum_i C[x[i]]\)
  - \((x, y) \in \{(0,0), (2,3), (3,2), (5,0), (0,5)\}\)
Some CP Constraints – 2

Global

- AllDifferent($x$)
- Cardinality: $| \{ x \in S : x = 3 \} | = 5$
- Path: $x$ defines a Hamiltonian cycle

Custom constraints

- Define your own by extending constraint classes
IP uses priorities and economic pseudocosts to direct branch-and-bound tree exploration scheme

CP requires explicit search algorithm

- Ex 1: Branch on variable with smallest domain
- Ex 2: Branch on variable with lowest costs

Exploiting problem structure leads to fastest solutions
Comparing IP and CP modeling

- Example: Pick two different items

<table>
<thead>
<tr>
<th>IP: Use binary variables and linear constraints</th>
<th>CP: Use general variables with logical constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Sigma_i x[i] = 1 )</td>
<td>( x \in {1, \ldots, n} )</td>
</tr>
<tr>
<td>( \Sigma_i y[i] = 1 )</td>
<td>( x \in {1, \ldots, n} )</td>
</tr>
<tr>
<td>( x[i] + y[i] \leq 1 ) for all ( i )</td>
<td>( x \neq y )</td>
</tr>
</tbody>
</table>

- Each has a natural formulation
  - IPs have more variables and constraints
  - CPs have large variable domains
Typical Applications

- **LP/MIP**
  - Planning models with complex economics where optimal solution is required

- **CP**
  - Operational models with complex operations where many good feasible solutions are required
Agenda

- About ILOG
- Brief introduction to constraint programming
- Exploring combinatorial optimization
- Exploring iterative optimization
- Discrete scheduling
Exercise 1: Solving a set covering IP

- Set of tasks
  - `enum Tasks ...;`

- Set of workers
  - `range Workers 1..nbWorkers;`

- A subset of workers are qualified to perform each task
  - `{Workers} qualified[Tasks] = ...;`

- Find cheapest set of workers who are qualified to cover all jobs
Completing exercise 1

- First, setup tutorial files and launch OPL Studio
  - `~gglockner/tutorial/setup`
  - `cd ~/ilog`
  - `opl`

- Select File > Open > Project and open `setcver.prj`
- Write objective function and covering constraint
- Run by selecting Execution > Run
- Stop by selecting Execution > Abort

- Note: Your personal ilog directory (e.g. `~/ilog`) contains all sample files and a shell script for OPL Studio
Exercise 2: Map coloring CP

- Set of countries
  - `enum` Country ...;
- Set of colors
  - `enum` Colors ...;
- Set of bordering pairs
  - `{adj}` Border = ...;
- Assign a color to each country so that no bordering countries have same color
Completing exercise 2

- Load `map.prj`
- Add logical constraint to require bordering countries to have different colors
- Run and use `continue` to find all solutions

Optional: Rewrite the problem to see if *this* map can be colored using fewer colors!
- Create binary indicator variable `colorUsed[i]`
- Add objective and constraint on `colorUsed` indicators
  - NB: Unlike MP, you can index a CP decision variable by another
- Change “solve” to “subject to”
- Save as `map2` since we will reuse old version for exercise 3
Exercise 3: Tracing the algorithm

- **The CP search procedure**
  - As ILOG Solver tries assignments, it propagates the effects to eliminate particular combinations.
  - Once it finds a solution, it backtracks through the search tree and generate alternate solutions.

- **OPL Studio can animate the search procedure**
  - A graphical tree represents the search tree.
  - A table shows the variable domains.
Completing exercise 3

- Load completed map coloring model (map.prj)
- Select **Execution > Browse Active Model**
- Right click on **Variables > color** and select **display domain**
- Select **Debug > Stop at Choice Point** and **Debug > Display Search Tree**
- Select **Execution > Next** to step through search procedure

- Make sure to uncheck **Debug > Stop at Choice Point** and **Debug > Display Search Tree** when finished
Demo 1: Advanced graphics

- OPL’s graphical drawing board lets you develop custom graphics for a search procedure
  - Lines, ellipses, rectangles, and polygons
  - Various colors are available
  - Text labels
  - Graphics are undone when backtracking in the search tree

- Try it! Run mapgr.prj
Exercise 4: Factory sequencing

- There are three car colors: red, blue, and gray
- Day is divided into periods
  - You can make one car each period
  - Changing colors requires one idle period
- You must meet daily production requirements
- A simplified supply chain application
Completing exercise 4

- Load `factory.prj`
- Add constraint to satisfy daily demands
  - Careful! Should be cumulative
- Add logical constraint for color changes
  - Useful logical operators:
    - \( = \) (equals), \( <> \) (not equals), \( \lor \) (or), \( \land \) (and), \( ! \) (not), \( \Rightarrow \) (implies)
Combinatorial Optimization

Improving performance

- CP models benefit from redundant constraints that reduce search space
  - Like IP cuts, these are implied by model structure

- Ex 1: No sequence of idle periods
  - For any two consecutive periods, at least one should not be idle

- Ex 2: Color change
  - If a period is idle, then the surrounding periods should have different colors
Improving performance – 2

- **Alternate formulation**
  - For each batch of cars of the same color, determine start time, duration, and the color
  - Focuses on runs rather than slots
  - Can use both formulations simultaneously!

- **More complicated, but more efficient**
  - Easier to constrain run length, etc.
  - Fewer variables with larger domains
  - Take advantage of scheduling algorithms in ILOG Scheduler
Overview

Agenda

- About ILOG
- Brief introduction to constraint programming
- Exploring combinatorial optimization
- Exploring iterative optimization
- Discrete scheduling
What is iterative optimization?

- **Solving a sequence of related models**
  - Try various simplified models
  - Decompose a large problem into subproblems
  - View model results under different scenarios
Exercise 5: Resolving

- **Chicken nuggets**
  - Sold in packages of 4, 6, and 9
  - You can buy 10 by combining 4 & 6
  - You cannot buy 7 by combining any of these boxes

- **What is the largest number you cannot make?**
  - Determine this iteratively, not by number theory! 😊
Completing exercise 5

- Load `nuggets.mod`
- Add constraints to determine if a total of nuggets can be created
- Load and run `nuggets.osc` OPLScript

Optional

- Try nuggets in sizes of 4,5,6 or 6,9,20
- Can you change the script to be more efficient?
Traveling Salesman Problem (TSP)

- **Canonical combinatorial optimization problem**
  - Create a tour that visits every city exactly once
  - Make the tour as cheap as possible

- **Common iterative solution method**
  - Solve an integer programming problem
  - If you receive sub-cycles, add constraints to break the cycles (subtours)
  - Repeat until you find a single tour
Demo 2: Solving the TSP

- Open OPLScript in tsp.osc
  - Solves the TSP as a series of integer programs
  - Animates subtours via the drawing board
- Try it!
  - You can change the data sets in the OPLScript
  - Note: Some data do not have graphics
Overview

Agenda

- About ILOG
- Brief introduction to constraint programming
- Exploring combinatorial optimization
- Exploring iterative optimization
- Discrete scheduling
Describing a scheduling problem

- **Modeling entities for scheduling problems**
  - Activity variables
    - Defined by start time, end time, and duration
    - Can be breakable
    - Various precedence constraints for activities
  - Resource constraints
    - Unary Resources
    - Discrete Resources
    - Reservoirs
    - State Resources with Transition Times

- **Combine scheduling with generic constraint programming elements**
Want to decide the sequence of tasks to build a house

```cpp
class Tasks { masonry, carpentry, plumbing, ceiling,
  roofing, painting, windows, facade,
  garden, moving }

int duration[Tasks] = [7, 3, 8, 3, 1, 2, 1, 2, 1, 1];
Activity a[t in Tasks](duration[t]);
```

Each task requires a certain amount of time to be completed

```cpp
task a[masonry] precedes a[ceiling];
task a[carpentry] precedes a[roofing];
task a[ceiling] precedes a[painting];
```

...
Each task requires one worker from a set

```csharp
enum Workers { Alan, Beth, Carl };  
UnaryResource worker[Workers];  
var int assign[Tasks,Workers] in 0..1;  
forall(t in Tasks, w in Workers)  
a[t] requires (assign[t,w]) worker[w];  
forall(t in Tasks) sum(w in Workers) assign[t,w] = 1;
```

Want to balance the workload

```csharp
Activity onsite[w in Workers];  
minimize max(w in Workers) onsite[w].duration
```

```csharp
forall(t in Tasks, w in Workers)  
assign[t,w]=1 => (onsite[w].start <= a[t].start  
& onsite[w].end >= a[t].end);  
sum(w in Workers) onsite[w].duration >=  
sum(t in Tasks) duration[t];
```
enum Tasks { masonry, carpentry, plumbing, ceiling, roofing, painting, windows, facade, garden, moving};
enum Workers {Alan, Beth, Carl};
int duration[Tasks] = [7,3,8,3,1,2,1,2,1,1];
scheduleHorizon = sum(t in Tasks) duration[t];

Activity a[t in Tasks](duration[t]);
Activity onsite[Workers];
var int assign[Tasks,Workers] in 0..1;
UnaryResource worker[Workers];

minimize max (w in Workers) onsite[w].duration
subject to {
    a[masonry] precedes a[carpentry];  a[masonry] precedes a[plumbing];
    a[masonry] precedes a[ceiling];    a[carpentry] precedes a[roofing];
    ...
    forall(t in Tasks, w in Workers) a[t] requires(assign[t,w]) worker[w];
    forall(t in Tasks) sum(w in Workers) assign[t,w] = 1;
}

forall(t in Tasks, w in Workers)
    assign[t,w]=1 => (onsite[w].start <= a[t].start
    & onsite[w].end >= a[t].end);
sum (w in Workers) onsite[w].duration >= sum (t in Tasks) duration[t];
};
Exercise 6: Scheduling

- Open and run the model `balhouse.mod`
- Look at the Gantt and resource charts
  - Double-click on the activities and resources

- Try adding a constraint that Beth cannot do the plumbing

- Try adding a constraint that Carl must be finished no later than day 13
  - Hint: Every activity has a start, end, and duration time