Airline Schedule Planning
1.963 (1.206J/16.77J/ESD.215J)

OPL Studio Tutorial
Date: 02/19/2003

1 OPL Studio

- OPL means Optimization Programming Language
- ILOG products most optimization tools used by academics and professionals (http://www.ilog.com)
- Based on CPLEX/CPLEX MIP technology
- As efficient as CPLEX/CPLEX MIP but take a little more memory
- Launching OPL Studio full version from a SUN Machine:
- The free trial version is restricted to problems with less than 300 constraints and 300 variables

2 Interface

- Console: error messages
- Solutions: objective value, solutions, printouts
- Solver: # of variables, # of constraints, memory, running time, etc.
- CPLEX: useful information for MIP such as number of nodes left in Branch-and-Bound tree, as well as the best solution found so far
- Help

3 Linear Programming

3.1 Simple problem: simple.mod

A model is a file with the suffix ***.mod*** (simple.mod).

- Data declaration
- Variable declaration

- Model
- Comment: how to include comments in a program?

/* This is a comment */

or

// This is a comment (to the end of the line)

3.1.1 Declaration (simple2.mod)

Basic Data Types:

- Declaring a real number
  - float f = 3.2;
  - float + f = 4.0; declare a non negative float
- Declaring an integer number:
  - int g = 4;
- Declaring an enumerated type: (gas.prj)
  - enum Products = { gas, chloride };

Variable declaration:

- Real variables:
  - var float x;
  - var float+ y;
- Integer variables:
  - var int w in 0..10;
  - var int z in 0..maxint; (maxint is the largest integer value accepted by OPL, maxint = 2^31)

Note: end each line with a semi-column

3.1.2 Data Structures

- Ranges (simple3.mod)
- Arrays,
  - One dimensional array (simple2.mod)
  - Initializing of multi dimensional arrays: (index, value) with #[ ]# instead of [ ] (simple3.mod and simple4.mod)

- Sets: (simple3.mod)
  - {rj} set_col = {i | i in rj: x[i] > 200};
- Structure:
  - Interesting when several attributes define an object
  - Interesting to model sparse transportation model

```
struct  Element{
int col;
int value;
};
{Element} A[1..4];
```

Example: $\begin{bmatrix} -1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

```
Element A = [{<1,-1>,  <4,-1>}, {<1,1> <2,-1>}, {<2,1> <3,-1>,<4,1>}, {<3,1>}];
Forall (i in 1..4)
    sum(el in A[i]) (el.value*x[el.col]) = b[i];
```

Do it for simple problem (simple4-2.mod)

3.2 Project

Isolating the data, creating a project (covering.prj, product.prj) made of:

- A model: covering.mod
- A data file: covering.dat

First, create your project then insert model and data file and finally save.

3.2.1 Inputs/Outputs

Reading inputs:

- From data file:
  - Int a[1..600000] < “a.dat”;
- From excel spreadsheet (simple_project1.prj), (simple_project1.prj)

- From user inputs:
  - float+ c << "Enter c:";

Displaying results and printing out to file:

- Display
- To excel spreadsheet (simple_project1.prj):
  - SheetWrite(sh, "G10:G11") (x);

3.3 Integer programming and mixed integer programming

3.3.1 Branch-and-Bound

Knapsack.prj;. We have a knapsack with a fixed capacity (an integer) and a number of items. Each item has an associated weight (an integer) and an associated value (another integer). The problem consists of filling the knapsack without exceeding its capacity, while maximizing the overall value of its contents

- Look at CPLEX ouput notebook for number of LP iterations
- How to change node selection strategy? Go to MIP Strategy
- Study time of execution (CPU)

3.3.2 Script

Script: To solve a sequence of models or/and perform sensitivity analysis on a project (script_simple1.osc)

Script can be used to implement column generation and Branch-and-Price

For an LP (simple1.mod)

- Reduced costs: x[d].rc;
- Optimal duals: ct[t].dual;

Note on optimality conditions of an LP

Reference: “The OPL Optimization Programming Language,” by P. Van Hentenryck, MIT Press