Getting Started with IBM ILOG CPLEX Optimization Studio

Rapid Development and Deployment of Optimization Models / Analytical Decision Support Solutions

Beta testing

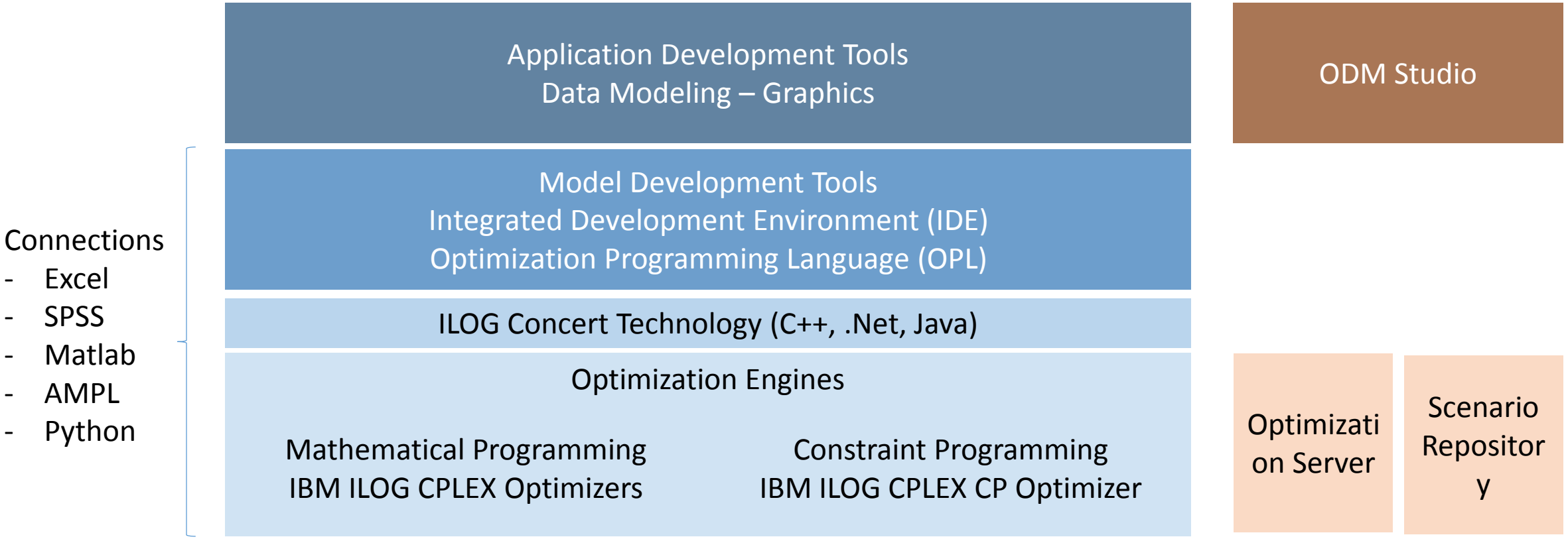
Pedro Amorim

Agenda | IBM CPLEX Optimization Studio

- The Optimization Programming Language (OPL)
  - Overview
  - Objective
  - Main Building Blocks
  - Optimization Engines
    - Mathematical Programming
    - Constraint Programming
- Developing OPL Models
- Exercise!
- Testing and Tuning OPL Models
- Deploying OPL Models

The Optimization Programming Language (OPL)

IBM ILOG
Cplex Optimization Studio



Simplify and Speed-Up Optimization Application Development

Business Expert

Business Analysts

Software Developer

Optimization Expert

CPLEX Studio's IDE, built on OPL and ILOG Concert Technology, dramatically reduces time-to-research/market for optimization applications

Inside IBM ILOG CPLEX Optimization Studio

- Integrated Development Environment (IDE)
  - Graphical editor for OPL models
  - Review data and solutions as tables
  - Menus/buttons to debug optimization
  - Online help for OPL syntax

Inside IBM ILOG CPLEX Optimization Studio

- Optimization Programming Language (OPL)
  - A compact language to represent optimization problems
  - Advanced types for data organization
  - Supports mathematical and constraint programming models
    - Linear and quadratic objectives and constraints
    - Real or integer variables
    - Integer variables, precedence and resource constraints
  - Connects to relation databases and Excel spreadsheets
  - OPL Script for data processing and iterative solving

Inside IBM ILOG CPLEX Optimization Studio

- OPL Script
  - An implementation of JavaScript
    - “Full-featured” scripting language that can be called inside OPL model files
    - Complements the declarative OPL syntax
    - For data pre-processing, solution post-processing, and flow control
    - Includes standard debugging tools: breakpoints, inspector
  - Applications
    - Compute model data from raw input data
    - Compute results data from solution values
    - Solve a sequence of related models
    - Implement decomposition strategies (column generation, custom cutting planes, etc.)

Inside IBM ILOG CPLEX Optimization Studio

- OPL interfaces
  - Built on ILOG Concert Technology
    - Links with IBM ILOG CPLEX Optimizers for MP and CP
  - Embed OPL models and scripts into applications
    - C++
    - Microsoft .NET
    - Java
    - APS.NET

Optimization Features of IBM ILOG CPLEX Optimization Studio

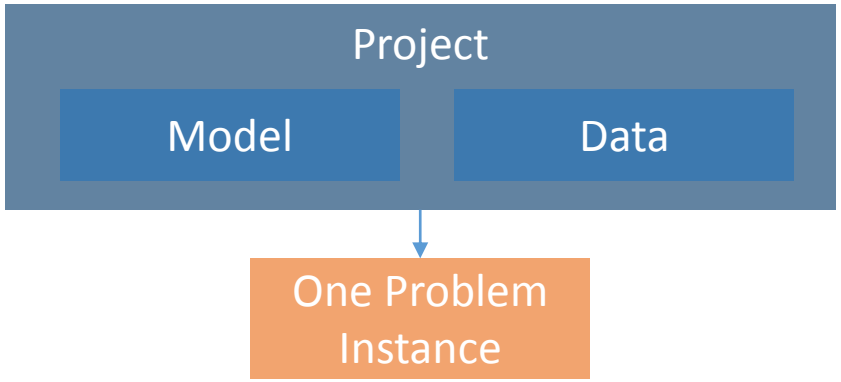
- Mathematical Programming
  - Solve linear and quadratic programs with CPLEX Simplex and Barrier Optimizers
  - Solve mixed integer linear and quadratic programs with CPLEX Mixed Integer Optimizer
  - Solve quadratically-constrained problems with CPLEX Barrier and Mixed Integer Optimizers
- Constraint Programming
  - Solve detailed scheduling problems with CPLEX CP Optimizer
  - Solve other combinatorial optimization problems with CPLEX CPL Optimizer

IBM ILOG CPLEX Optimization Studio provides access to all CPLEX Simplex, Barrier, Mixed Integer and CP Optimizer algorithm settings

Developing OPL Models

What are Models?

- A data – independent abstraction of a (business) problem
- A mathematical representation of key relationships in the problem
  - Decisions – Decision Variables
  - Goals and key performance indicators (KPIs) – Objective Functions
  - Limit, requirements, recipes - Constraints
- OPL lets you write a mathematical representation of your problem separately from you data



A Production Planning Example

- A manufacturer wants to sell a product
- The product can be made either
  - Inside the factory
    - Scarce resources are used
    - Cost per unit to manufacture
  - Outside the factory
    - Higher cost per unit to purchase
  - All demand must be satisfied
  - Goal/Objective: minimize total cost

Data Declarations

- Index sets for products and resources

```
{string} Products = ...;
{string} Resources = ...;
```

- Recipe – units required of each resource per unit of each product

```
float Consumption[Products][Resources] = ...;
```

- Amount of each resource available

```
float Capacity[Resources] = ...;
```

- Amount of each product required

```
float Demand[Products] = ...;
```

- Cost per unit of inside and outside production

```
float InsideCost[Products] = ...;
float OutsideCost[Products]  = ...;
```

Products Could be Jewely

- Index sets for products and resources

```
Products =  { “rings", “earrings" };
Resources = { “gold", “diamonds" };
```

- Recipe

```
Consumption =   gold    diamonds
                [ [3.0, 1.0],   rings
                [2.0, 2.0] ];   earrings
```

- Amount of each resource available

```
Capacity = [ 130, 180 ];
```

- Amount of each product required

```
Demand = [ 100, 150];
```

- Cost per unit of inside and outsider Production

```
InsideCost = [ 250, 200];
OutsideCost  = [ 260, 270 ];
```

Products Could be Pasta

- Index sets for products and resources

```
Products = { "kluski", "capellini", "fettucine" };
Resources = { "flour", "eggs" };
```

- Recipe

```
Consumption =           flour    eggs
                        [ [0.5, 1.0],   kluski
                        [0.4, 0.4],    capellini
                        [0.3, 0.6] ];   fettucine
```

- Amount of each resource available

```
Capacity = [ 20, 40 ];
```

- Amount of each product required

```
Demand = [ 100, 200, 300 ];
```

- Cost per unit of inside and outsider Production

```
InsideCost [ 0.6, 0.8, 0.3 ];
OutsideCost  = [ 0.8, 0.9, 0.4 ];
```

The Model is the same

```
{string} Products = ...;
{string} Resources = ...;

float Consumption[Products][Resources] = ...;
float Capacity[Resources] = ...;
float Demand[Products] = ...;
float InsideCost[Products] = ...;
float OutsideCost[Products]  = ...;

dvar float+ Inside[Products];
dvar float+ Outside[Products];
```

Data Definition

Decision Variables

The Model is the same

```
minimize
  sum( p in Products )
    ( InsideCost[p] * Inside[p] + OutsideCost[p] * Outside[p] );

subject to {
  forall( r in Resources )
    ctCapacity:
      sum( p in Products )
        Consumption[p][r] * Inside[p] <= Capacity[r];

  forall(p in Products)
    ctDemand:
      Inside[p] + Outside[p] >= Demand[p];
}
```

Objective Function

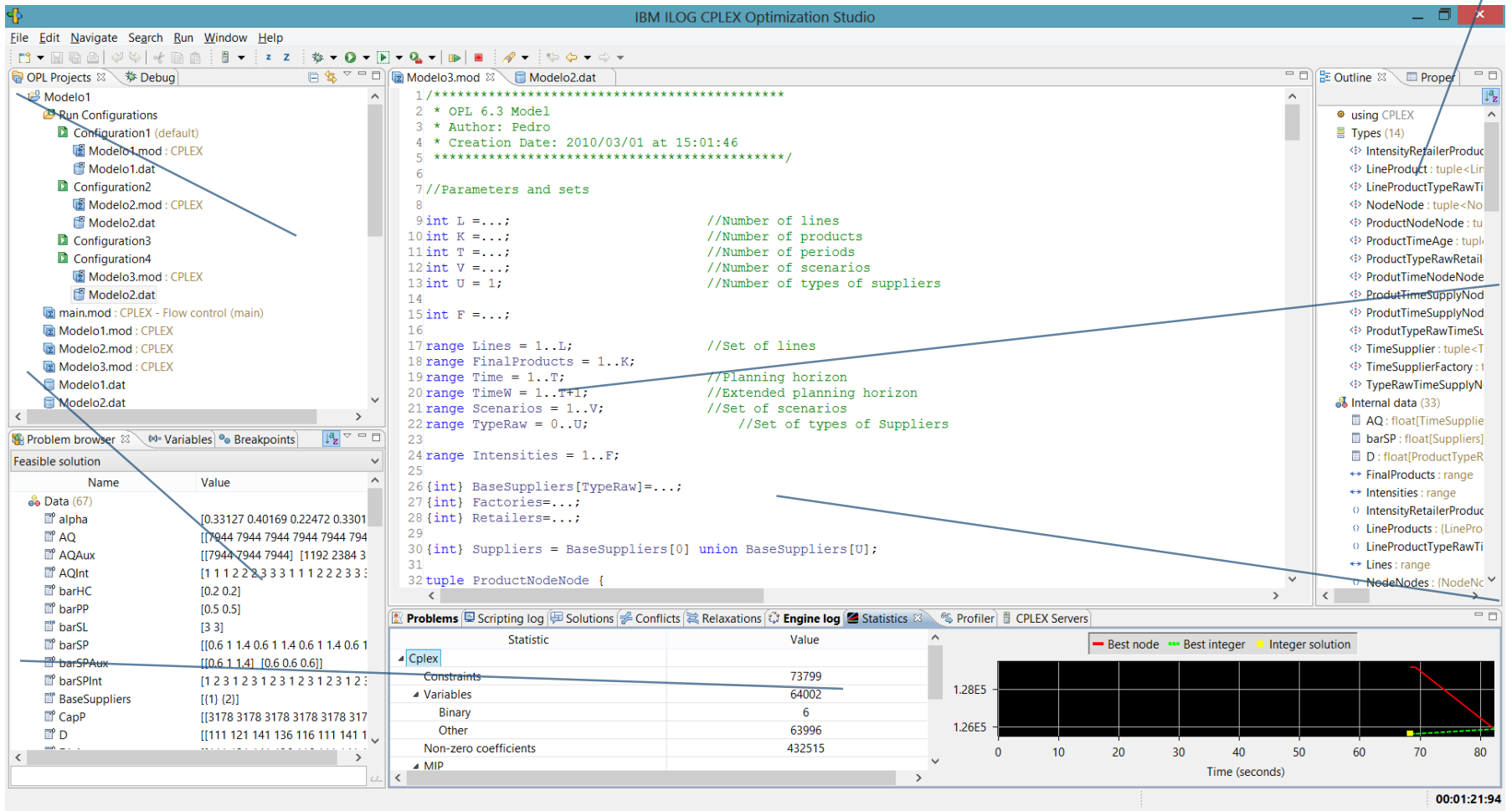
Constraints

IBM CPLEX Optimization Studio: IDE

Project Navigator

Problem Browser and Solution Inspection

Inspector/ Output Area



Outline

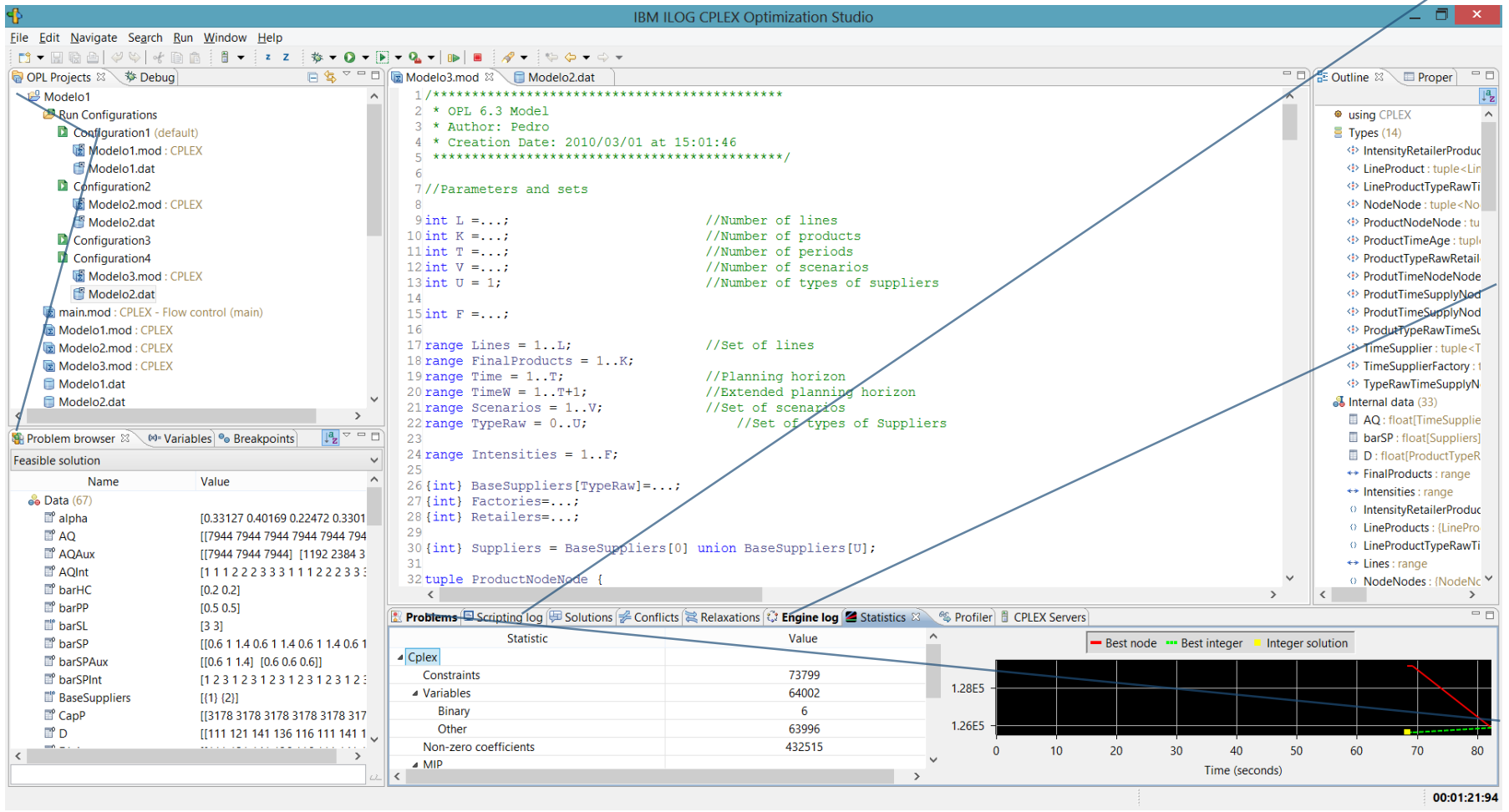
OPL Statements

Editing Area

IBM CPLEX Optimization Studio: IDE

Running a configuration

A run configuration = A variation of a project for execution purposes
<> .mod
<> .dat
<> .ops



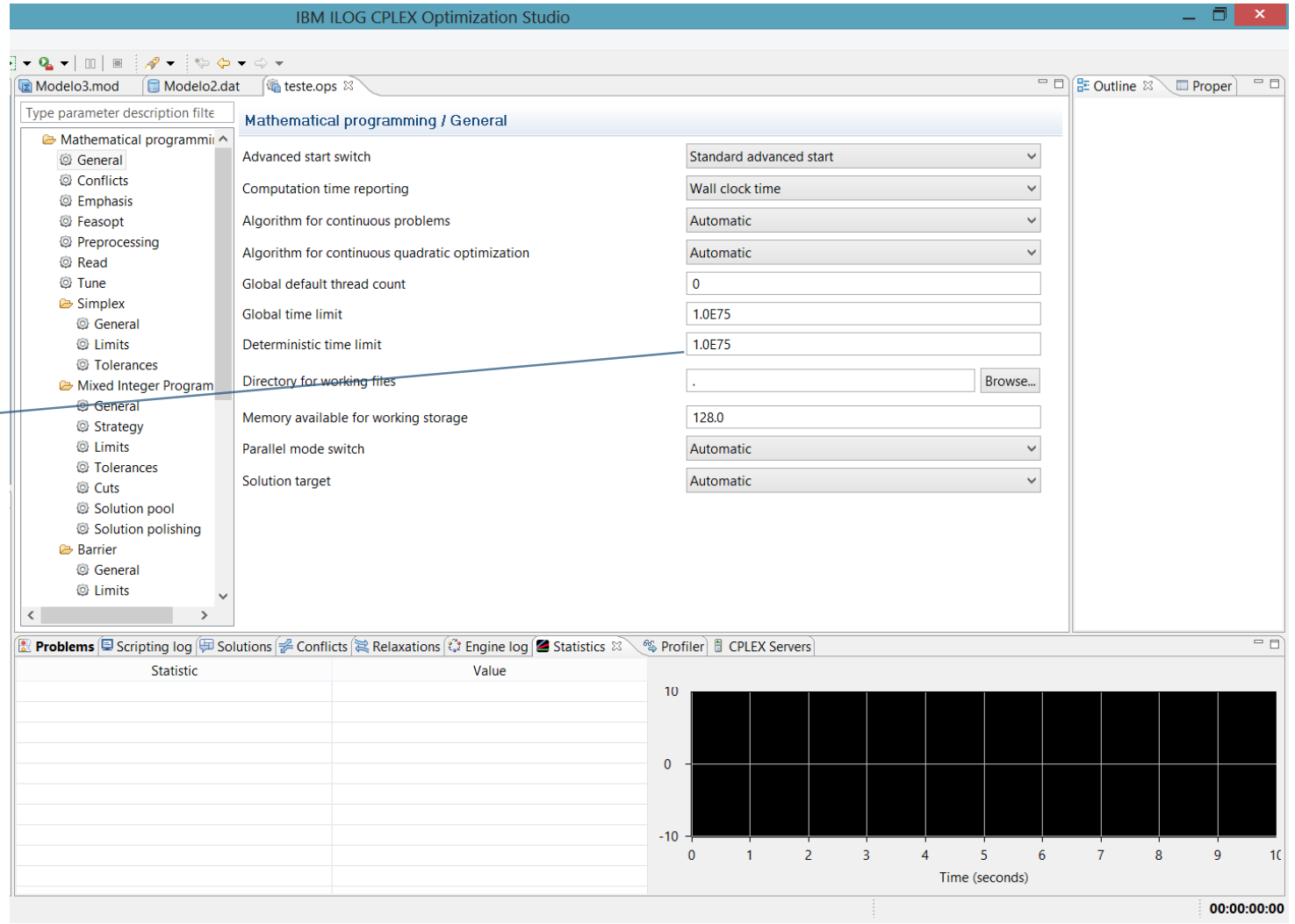
Script Output

Engine Log!

Main Debugging Reports

IBM CPLEX Optimization Studio: IDE

Settings File



The Optimization Programming Language (OPL)

- Primitive Types

```
int, float, string
```

- “Tuples”

```
tuple Route {
        string o; /*origin*/
        string d; /*destination*/
        string p; /*product*/
}
```

- Sets

```
setof(string) cities = …;
{Route} routes = …;
```

- Arrays

```
float cost[routes]=…;
```

The Optimization Programming Language (OPL)

- Arrays

```
range Trucks = 1..maxTrucks;
```

- Decision Variable – “dvar”
  - Int, float, boolean
  - Index on Range or Set

```
dvar float+ shipments[routes] in 0..capacity[r];
dvar int nTrucks[Trucks];
dvar boolean open[cities];
```

- Decision Expression - “dexpr”
  - For readability and goal programming in ODM Enterprise

```
dexpr shippingCost = sum (r in routes) cost[r]*shipments[r];
```

Sparse Data Structures

- Given a set of cities

```
setof(string) Cities = {“PIT” “FRA” “WIN” “LAF” };
```

- Routes are represented as

```
tuple Route {string o; string d; string p};
setof(Route) routes = {<“FRA”, “WIN”, “rings”>, <“FRA”, “LAF”, “rings”>, <“PIT”, “WIN”, “rings”>};
```

- The origin cities are

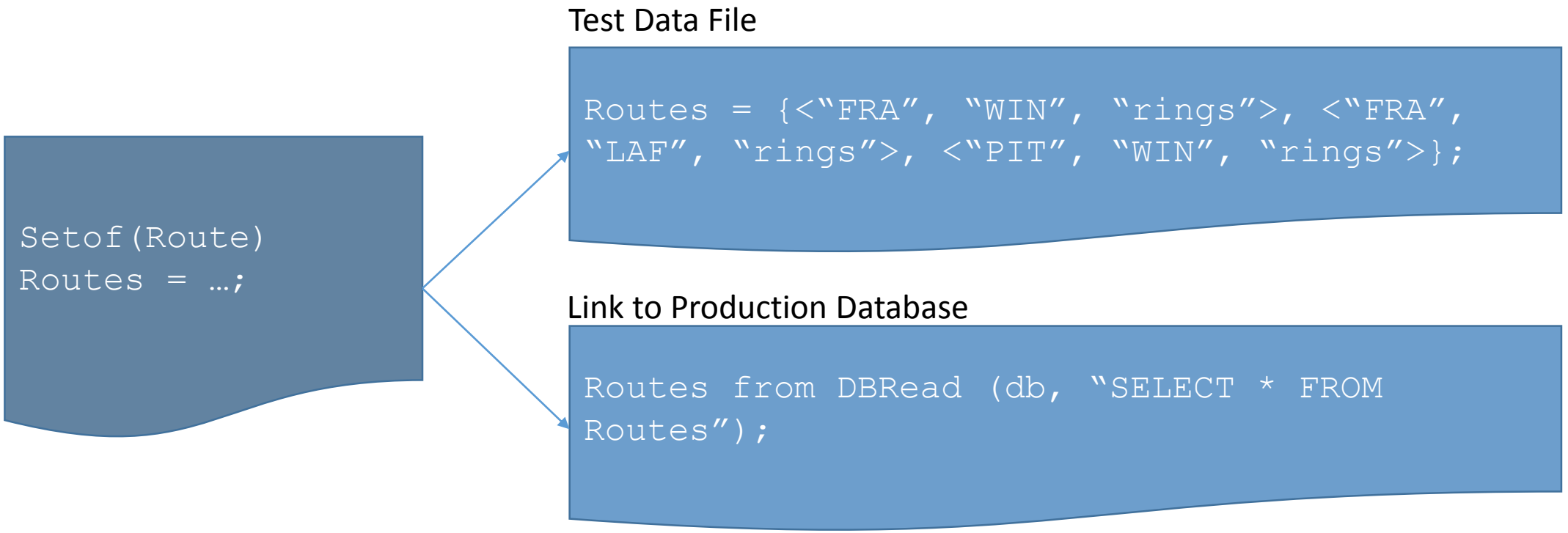
```
setof(string) origins = {o | <o,d,p> in routes};
```

- Define a decision variable array over routes

```
dvar float+ shipments[routes];
```

Model/Data Separation

- Data access is completely separate from the model



- No changes to model to switch to production data

Reading External Data Sources

- From a spreadsheet

```
SheetConnection sheet("warehouse.xls);
routes from SheetRead(sheet, "Data!A2;C4");
cities from SheetRead(sheet, "cities");
```

- From a database

```
DBConnection db("acess", "warehouse.mdb");
routes from DBRead(db, "SELECT * FROM Routes");
cities from DBRead(db, "SELECT Origin FROM Routes Union
SELECT Destination FROM Routes");
```

Besides Reading it is also possible to Write

- From a spreadsheet

```
SheetConnection sheet("warehouse.xls);
shipments to SheetWrite(sheet,   "Data!D2;D4");
```

- To a database

```
DBConnection db("access", "warehouse.mdb");
results to DBUpdate(db, "INSERT INTO  Shipments
(origin,   destination, product,      quality) values
(?,?,?,?)");
```

Detailed Scheduling Features

- Intervals
  - Models an event that occurs over time

```
dvar interval job[t in tasks] in t.earlyStart .. t.lateEnd size
t.duration;
```

- Cumulative Functions
  - Models Resource Usage

```
cumulFunction workersUsage = sum(t in tasks) pulse(job[t],1);
```

- Other Scheduling Specific Constructs
  - Sequences, Intensities, etc.

(Refer to IBM's White Papers)

Data Pre-processing

```
//OPL
float long[cities]=…;
float latt[cities]=…;
float freightRate[products]=…;

float Cost[routes];

//Script
execute INITIALIZE {
var longDiff;
var lattDiff;
for(var r in routes) {
longDiff = long[r.o]-long[r.d];
lattDiff = latt[r.o]-latt[r.d];
Cost[r] = freightRate[r.p]*Opl.sqrt(Opl.pow(longDiff,2)+Opl.pow(lattDiff,2));
}
```

Flow Control – Iterative Solving

```
main {
        var prj = thisOplModel;
        var mod = prj.modelDefinition;
        var dat = prj.dataElements;
        var rate = dat.freighRate["rings"];
        for (var i=0; i <= 10; ++i) {
                cplex.clearModel();
                prj = newIloOplModel(mod,cplex);
                prj.addDataSource["rings"]=rate;
                prj.generate();
                cplex.solve();
                writeln("Freight rate for rings ", rate, " , Objective function: ",
cplex.getObjValue());
                rate*= 1.1;
        }
}
```

Initialize project

Reset model

Solve & reset value

Solution Post-processing

```
//Script
execute OUTPUT_RESULTS {
   var file = new IloOplOutputFile("RiskNeutral.txt", true);

//output solution

file.writeln("ObjValue \t",cplex.getObjValue());
}
```

Exercise

Blending problem

- They involve blending several resources or materials to create one or more products corresponding to a demand.
- Mixed integer-linear programs are linear programs in which some variables are required to take integer values.

- The metal blending example involves mixing some metals to form an alloy. The metal may come from several sources: in pure form, from raw materials, as scraps from previous mixes, or as ingots. The alloy must contain certain amounts of the various metals, as expressed by a production constraint specifying lower and upper bounds for the quantity of each metal in the alloy. Each source has a cost and the problem consists of blending from the sources while minimizing the cost and satisfying the production constraint. Similar problems arise in other domains, for example, the oil, paint, and food-processing industries.

- Indices
  - m – Metals
  - r – Raw Materials
  - s – Scraps
  - i – Ingots
- Parameters
  - CM_m – Cost of Metals (€/ton)
  - CR_r – Cost of Raw-Material (€/ton)
  - CS_s – Cost of Scraps (€/ton)
  - CI_i – Cost of Ingots (€/ton)
  - MAX_m – Maximum percentage of a given Metal
  - MIN_m – Minimum percentage of a given Metal
  - PR_{mr} – Percentage that Raw Material r has of Metal m
  - PS_{ms} – Percentage that Scrap s has of Metal m
  - PI_{mi} – Percentage that Ingot i has of Metal m
  - DA – Demand for the Alloy (ton)

- Decision Variables
  - pq_m – used quantity of pure Metal (ton)
  - rq_r – used quantity of Raw Material (ton)
  - sq_s – used quantity of Scrap (ton)
  - iq_i – number of ingots used (units) (1 unit = 1 ton)
  - fq_m – final quantity of Metal (ton)

- Objective Function
  - $\min \sum_m CM_m pq_m + \sum_r CR_r rq_r + \sum_s CS_s sq_s + \sum_i CI_i iq_i$
- Constraints
  - $MIN_m DA \leq fq_m \leq MAX_m DA \quad \forall m$
  - $fq_m = pq_m + \sum_r PR_{mr} rq_r + \sum_s PS_{ms} sq_s + \sum_i PI_{mi} iq_i \quad \forall m$
  - $\sum_m fq_m = DA$
  - Non-negativity.

Exercise Steps

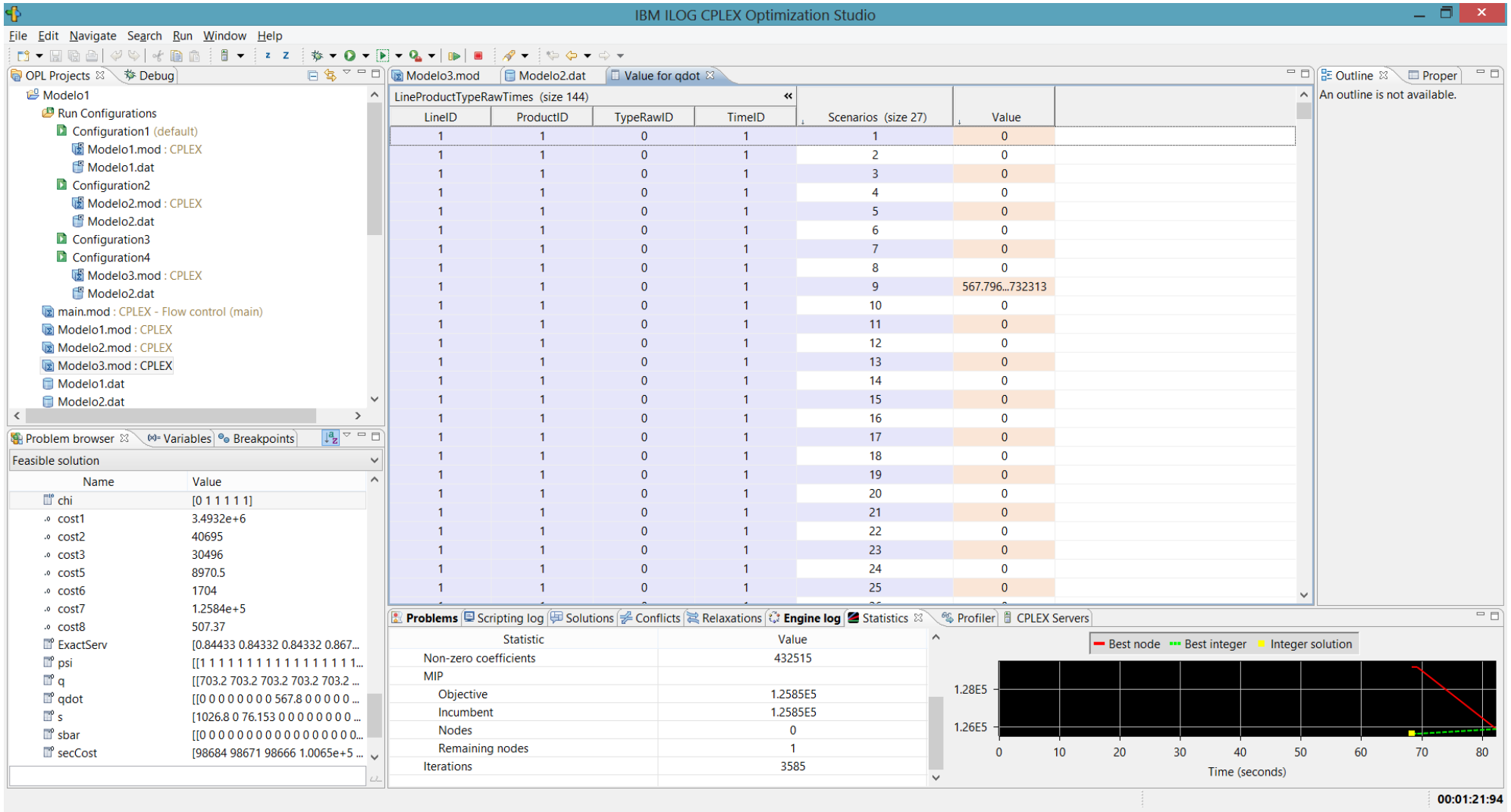
a) Write the Model in OPL
b) Could you incorporate the first constraint in the definition of the decision variable?
c) Get the .mod and .dat from \\examples
d) Run the Model
e) Write a Script to output the solution to a .txt
f) Link your project with an excel spreadsheet
g) Write automatically the quantities of the different metals in the spreadsheet

Testing and Tuning OPL Models

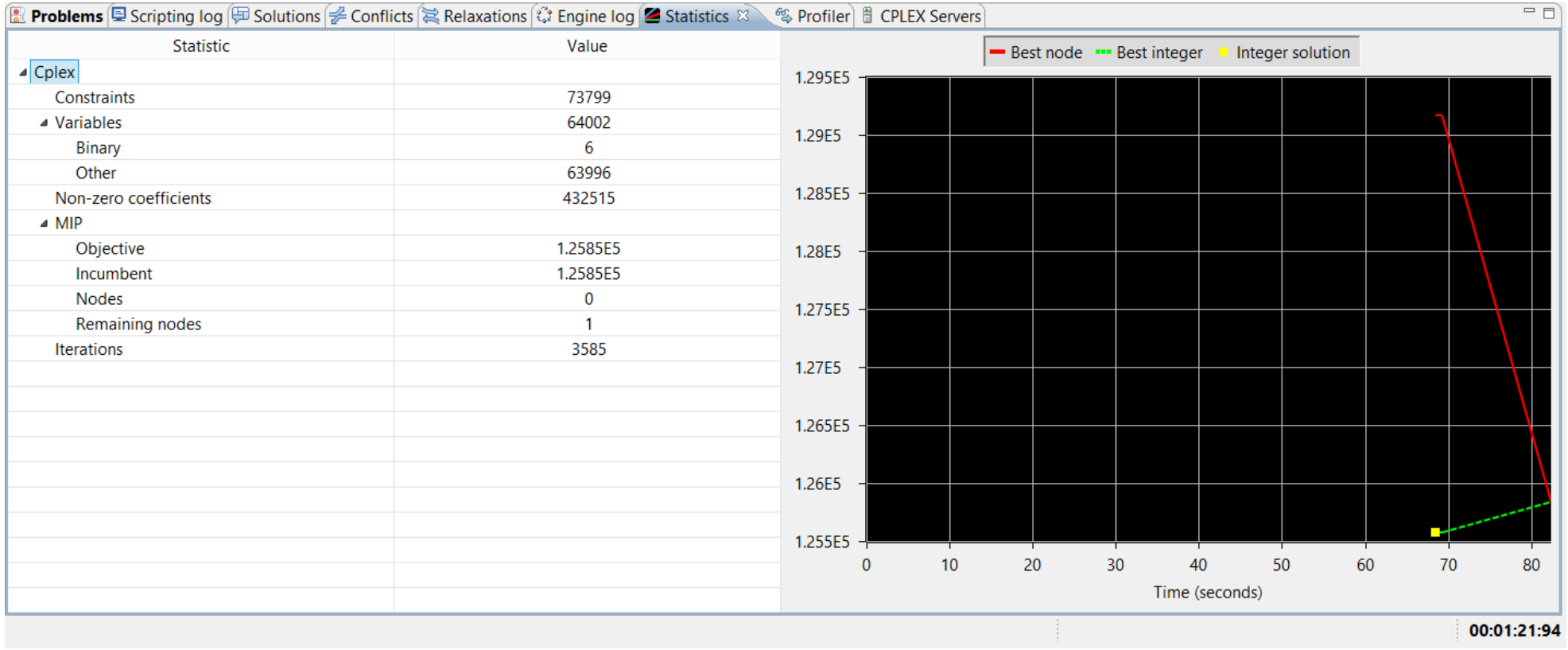
Debugging Features of IBM CPLEX Optimization Studio

- Model development
  - Tabular views of data and decisions variables
  - Iterative constraint expansion
- Execution tracing
  - Breakpoints
  - Solve, pause, and continue
  - Graph MIP progress
  - View CPLEX Optimizer log
- Export for testing
  - Dump data to flat file
  - Create .mps and .lp model files

Tabular views of data, decision variables and constraints



Execution Tracing



- Can pause execution short of optimality to browse the intermediary solution

Deploying OPL Models

Rapid Application Deployment

- Develop your model in IBM ILOG CPLEX Optimization Studio's IDE, maintaining model/data separation
- Incorporate model via OPL Interfaces
  - Use C++, Java, C#, Visual Basic.NET, ASP, JSP (Next Session!)
  - Integrate external data
- Link with solvers
  - Math programming: IBM ILOG CPLEX Simplex, Barrier or Integer Optimizers
  - Constraint programming: IBM ILOG CPLEX CP Optimizer

Ease communication with companies with IBM ILOG ODM Enterprise

A deployment platform offering the quickest path from OPL models to rapidly create business-user friendly applications

- Flexible tabular and chart views of data and solutions out-of-the-box
- Access multiple data sources
- Multiple objectives, goal programming
- Modify data, constraints, and business rules
- Relax constraints and diagnose infeasibilities
- Create and compare what-if scenarios
- Collaborate among multiple users
- Support decision processes with OR, IT, planner, and reviewer roles
- Use optimization server to handle intense, distributes computations requirements

KEY Features of IBM CPLEX Optimization Studio

- Powerful, graphic IDE
  - Entering models and data
  - Organizing projects
  - Debugging
  - Visualizing data and solutions
  - Controlling optimization
- Expressive OPL modeling and scripting language
  - Linear, Quadratic, and Constraint Programming
  - Continuous and integer decision variables
  - Detailed scheduling constructs
- Integrated solver technologies
  - IBM CPLEX Optimizers
- Database and Spreadsheet access
- OPL Interfaces accelerate model evaluation and deployment

Getting Started with IBM ILOG CPLEX Optimization Studio

Rapid Development and Deployment of Optimization Models / Analytical Decision Support Solutions

Beta testing

Pedro Amorim