Modeling with Xpress

FICO™ Xpress Optimization Suite whitepaper

Last update 26 May, 2009

Modeling with Xpress

Bob Daniel

Fair Isaac Corporation, Leam House, Leamington Spa CV32 5YN, UK
`http://www.fico.com/xpress`

26 May, 2009

Abstract

A paper discussing the various options open to the application developer for building and solving models using FICO^{TM} Xpress.

Contents

1 Introduction

Model development is perhaps the most intellectually challenging part of practical mathematical programming. But having once developed an algebraic model of the situation, you are faced with deciding how to implement the model inside some larger computer system. The primary considerations you will have are, typically and in no particular order:

- verify model correctness

- ease of model maintenance and modification

- algorithmic considerations

- data access and manipulation

- model execution speed

- speed to market

This paper discusses the various options open to you, the application developer, for building and solving models using XpressMP. In particular, it assesses their relative strengths and weaknesses under these criteria of using three approaches

- building models in Mosel, the modeling and optimization environment and language, and deploying them using Mosel's libraries and Xpress-Optimizer's library.

- building and deploying models in your application using the Xpress Builder Library BCL, together with Xpress-Optimizer's library.

- building and deploying models in the native language of your application and loading a complete problem instance directly into Xpress-Optimizer's library.

In the Appendix we take a small model and show it in Mosel's modeling language and using BCL (with C, C++ and Java).

2 Mosel's Language

Mosel's modeling language has been designed to be as close as possible to the algebraic formulation of the model, subject to the limitations of the characters available on the keyboard. For instance, the algebraic constraints

$$\forall j \in \{1, \ldots, NJ\} : \sum_{t=1}^{NT-D_j+1} t\delta_{jt} = s_j$$

are expressed in Mosel as

```
forall(j in 1..NJ) sum(t in 1..NT-D(j)+1) t*delta(j,t) = s(j)
```

A sophisticated model can be constructed using a simple syntax that is easy to understand and quick to implement. Here is part of an example.

```
declarations
 NT = 36; NF = 6; NP = 10          ! Time periods; factories; products
 T = 1..NT ; F = 1..NF; P = 1..NP  ! Useful ranges
 MXMK: array(F) of real            ! A real table
 YES: array(F,T) of boolean        ! A table of Boolean
  ...
 make: array(P,F,T) of mpvar       ! Decision variables
 open: array(F,T) of mpvar
end-declarations

! Here come some constraints
 Profit:= -sum(p in P,f in F, t in T) MCOST(p,f)*make(p,f,t)
 forall(f in F, t in T) sum(p in P) make(p,f,t) <= MXMK(f)*open(f,t)
 forall(f in F, t in T | YES(f,t)>0) open(f,t) is_binary
```

Other features include index sets, powerful integer programming constructs such as partial integers, semi-continuous variables, special ordered sets and model cuts, and integer programming directives. In addition, there is a very powerful programming language, with looping, selections, ranges and sets, and all the constructs of a full programming language.

2.1 Software Tools

The Xpress product suite comprises a sophisticated set of tools to develop and maintain models. The best tool for developing and debugging models is Xpress-IVE, the integrated modeling and optimization development environment for Windows.

Mosel is a modeling and optimizing environment that is suited to model development on all computer platforms. Models can be "called" from programming languages, such as VB, C/C++, Java, or C#, and embedded within applications using the Mosel libraries. So Mosel has different parts/interfaces that correspond to the different possible uses of the software: a command line version for standalone use; libraries enabling integration and use of existing algorithms written in C/C++ *etc.*; and the language underlying Xpress-IVE.

2.2 Clarity and Simplicity

As the model is close to its algebraic representation, models are typically very short and understandable. The modeling language can rapidly be learned and applied, allowing models to be written, understood and modified quickly and easily. The model is read from a simple text file, which can be modified and integrated into an application without having to rebuild the application. Mosel can compile the model, which is then executable on any platform.

2.3 Security

Using Mosel makes the application easy to build, understand, modify and maintain. In many applications the developer wishes to prevent the end user from reading or modifying the model files. For example, there may be commercial reasons related to intellectual property or secrecy, or there may be concerns that end-user modifications could adversely affect quality or reliability. Mosel's compiled (BIM) files hide the model from the end user completely.

2.4 Data Access and Manipulation

The methods for manipulating data within Mosel are as powerful, if not more so, as those provided by a high level programming language. Mosel provides a very high level data interfacing functionality, allowing common data import and export tasks to be accomplished with a single line in the model.

So Mosel provides different ways of accessing data, including Mosel's own format files, which are very easy to set up; freely formatted text files; data held in memory or generated by other applications; and via the (additional) ODBC module, access to any database that has an ODBC interface and to Microsoft Excel spreadsheets. New data sources and formats can be freely defined by the user.

2.5 Optimization

To solve linear, integer and quadratic programming problems, Mosel uses the Xpress-Optimizer library. Extension modules provide access to the other solvers of the Xpress suite: Xpress-SLP for solving nonlinear programming problems, Xpress-SP for stochastic models, and Xpress-Kalis for using a finite domain and floating point constraint solver. Mosel can also be interfaced to other

solvers, for instance Tabu Search, for specialized solving techniques.

2.6 Summary

Verify model correctness	easy
Maintenance and modification	easy
Algorithmic considerations	internal
Data access and manipulation	high level
Model execution speed	potentially slightly slower than the other methods
Speed to market	fast
Why use it?	good for getting 95% applications to market quickly

3 BCL

The philosophy behind BCL is that from within a high level programming language (C, C++, Java, C, or VB) you use a library designed for building matrices. Matrices can be constructed in a very flexible manner, bit by bit, in no pre-ordained order.

Let's see how our equation might be written in C. We first get all the variables onto the left of the constraint:

$$\forall j \in \{1, \ldots, NJ\} : \sum_{t=1}^{NT-D_j+1} t\delta_{jt} - s_j = 0$$

Then the code fragment might be, where the BCL functions are highlighted:

```
for(j=0; j<NJ; j++) {
  ctr = XPRBnewctr(prob, "C3", XPRB_E);
  for(t=0; t<(NT-D[j]+1); t++)
     XPRBaddterm(ctr, delta[j][t], t+1);
   XPRBaddterm(ctr, s[j], -1);
}
```

We have omitted the C declarations. In the context of a problem prob, the call to `XPRBnewctr()` creates a (pointer to a) new constraint; and `XPRB_E` specifies that it is an equality constraint. At this point the constraint is empty, but `XPRBaddterm()` adds a term with value `t+1` to the constraint for variable `delta[j][t]`. We use the looping facilities of C to do this for the desired values of t within the loop over all the possible j values. Finally, we use `XPRBaddterm()` again to add the variable `s[j]` to the constraint with a coefficient of -1. By default, the right hand side of the constraint is 0.

3.1 Security

BCL comes as a set of routine calls, so the model is completely hidden inside the high level program. If that is compiled, then the model is not accessible by the end-user.

3.2 Data Access and Data Manipulation

You have all the data handling facilities provided by your programming language. For ODBC this may not be easy but as long as you are prepared to program then you can access any source. Moreover, you can perform any manipulations or transformations on the data you have acquired. You can do verification tests, write sophisticated diagnostics, *etc.*

3.3 Model Flexibility

It is much harder to make models flexible when they have been encoded in a programming language. Suppose, for instance, that you have built, debugged and deployed a Mosel model to your end users. And then you want to modify it, perhaps adding a new class of constraints. All you have to do is to email them the modified model file. But if you are using a programming interface, then you have to rebuild your application, spend a lot more time testing it, and then send out a much larger executable.

3.4 Summary

Verify model correctness	quite easily verified (harder than Mosel, easier than XOSL)
Maintenance and modification	harder than Mosel, easier than XOSL
Algorithmic considerations	easy to build algorithms and exploit structure
Data access and manipulation	native programming language; some high level BCL intrinsic
Model execution speed	probably faster than Mosel, probably the same as XOSL
Speed to market	faster than XOSL, slower than Mosel
Why use it?	complete flexibility to build and modify model within your application

4 Optimizer Library

The Xpress-Optimizer Library interface is the lowest level into the Optimizer. You set up the data structures that the optimizer requires in your programming language, and then pass it to the library. The library is then instructed to do various things—for instance, to solve the model—and then you can access the optimal values from your program.

A clue to the (necessary) complexity of the interface can be obtained by looking at the specification of the routine to load an integer programming problem into the library. It has 23 arguments!

```
int XPRSloadglobal(XPRSprob prob, char *probname, int ncol,
  int nrow, char *qrtype, double *rhs, double *range, double *obj,
  int *mstart, int *mnel, int *mrwind, double *dmatval, double *dlb,
  double *dub, int ngents, int nsets, char *qgtype, int *mgcols,
  double *dlim, char *qstype, int *msstart, int *mscols,
  double *dref);
```

where you have to specify

prob   problem pointer

probname   name for the problem

ncol   number of structural columns (variables)

nrow   number of rows (constraints)

qrtype   row types (equality, less-than-or-equal-to, etc)

`rhs`	RHS coefficients of the rows
`range`	range values for range rows
`obj`	objective function coefficients
`mstart`	offsets in the mrwind and dmatval arrays of the start of the elements for each column
`mnel`	number of elements (*i.e.*, coefficients of the variables) for each column
`mrwind`	row indices for the elements in each column
`dmatval`	element values
`dlb`	lower bounds on the columns
`dub`	upper bounds on the columns
`ngents`	number of binary, integer, semi-continuous and partial integer variables
`nsets`	number of special ordered sets (S1 and S2)
`qgtype`	integer variable types (binary, integer, etc)
`mgcols`	column indices of the integer variables
`dlim`	bounds for the partial integer variables and semi-continuous variables
`qstype`	Special Ordered Set types (S1, S2)
`msstart`	offsets into the mscols and dref arrays indicating the start of the sets
`mscols`	the columns in each set
`dref`	reference row entries for each member of the sets

One major difficulty is that you must specify the non-zeros in the constraint matrix by going down the first column, then the second column, and so on. Thus if constraints are added or dropped from the formulation, the program has to be modified in a major way.

The only real advantage of "modeling" in the library over modeling in BCL is that the library is very marginally quicker, as BCL itself has to massage the data into the `loadglobal()` format before it passes data across to the Optimizer library. This takes very little time, but obviously, if you do it in your program, interfacing directly to the Optimizer library, that small overhead is avoided. (Of course, you can use the Optimizer library's very powerful features with a matrix which has been created with BCL, so nothing except a little speed is lost by modeling with BCL.)

We are not going to give you our prototypical equation written directly for the Optimizer library interface, for two reasons. The first is that the interface is activity (column) oriented. The second is that it is a nightmare.

Traditionally optimization subroutine libraries like the Optimizer library have provided the only means to write efficient applications using optimization, and such libraries are widely used for that purpose. But the Mosel language and BCL offer real advantages for building and manipulating problems, and near zero cost in terms of speed of application.

4.1 Summary

A perhaps prejudiced view of the advantages/disadvantages of the direct Optimizer library interface is:

Verify model correctness	very hard
Maintenance and modification	extremely difficult
Algorithmic considerations	easy to build algorithms
Data access and manipulation	native programming language
Model execution speed	probably the fastest
Speed to market	slowest
Why use it?	most efficient but lose easy model development and maintenance

5 A Complete Model

5.1 Mosel

Here is a complete Mosel model.

```
model sched
 uses "mmxprs"              ! Xpress-Optimizer is used
 declarations
  NJ  =  4 ; NT = 10        ! Number of jobs  / time limit
  J = 1..NJ; T = 1..NT      ! Useful ranges
  D: array(J) of integer    ! Table for durations of jobs
  s: array(J) of mpvar      ! Start times of jobs
  delta: array(J,T) of mpvar ! Binaries for start times
  z: mpvar                  ! Maximum completion time (makespan)
 end-declarations

 D:: [3, 4, 2, 2]           ! Durations of jobs

 forall(j in J) s(j) <= NT-D(j)+1 ! Interval for start times
 forall(j in J, t in 1..NT-D(j)+1 ) delta(j,t) is_binary ! Binaries

! The constraints
 forall(j in J) do
    ! Calculate maximum completion time of all jobs
   C1(j):=  z >= D(j) + s(j)
    ! Relation linking start times of jobs with corresponding binaries
   C3(j):= SUM(t in 1..NT-D(j)+1) t*delta(j,t) = s(j)
    ! One start time for each job
   C4(j):= SUM(t in 1..NT-D(j)+1) delta(j,t) = 1
 end-do

  ! Precedence relation between two pairs of two jobs
 C2_31:= s(3) >= D(1) + s(1)     ! 3 must follow 1
 C2_41:= s(4) >= D(1) + s(1)     ! 4 must follow 1

  ! Objective function to be minimized
 minimize( z )

 writeln(" Min makespan is ", getobjval)
 forall(j in J) writeln(" Job ", j, " starts at time ", getsol(s(j)))

 end-model
```

5.2 Mosel Runtime

In Mosel runtime, the same model can be used. Here is a typical application in C that loads the compiled model, solves the MIP, and prints a little bar chart. We assume that the compiled model is held in a file sched.bim.

```c
#include <stdio.h>
#include "xprm_rt.h"

int main(int argc, char **argv)
{
 XPRMmodel mod;
 XPRMalltypes rvalue;
 XPRMarray varr, darr;
 XPRMmpvar s;
 int indices[1], result, nt, t, D;

 XPRMinit();                         /* Initialize Mosel */

 mod=XPRMloadmod("sched.bim", NULL);  /* Load a BIM file */

 XPRMrunmod(mod, &result, NULL);      /* Run & optimize the model */

 printf("\nMinimum makespan %g\n", XPRMgetobjval(mod));

 XPRMfindident(mod, "s", &rvalue);    /* Get the model object 's' */
 varr = rvalue.array;
 XPRMfindident(mod, "D", &rvalue);    /* Get the model object 'D' */
 darr = rvalue.array;
 XPRMfindident(mod, "NT", &rvalue);   /* Get the model object 'NT' */
 nt = rvalue.integer;

/* Print a little bar chart */
 printf("Job   Time:1234567890\n");
 XPRMgetfirstarrentry(varr, indices); /* Get 1st entry of array varr */
 do
 {
  XPRMgetarrval(varr,indices,&s);     /* Get a variable from varr */
  XPRMgetarrval(darr,indices,&D);     /* Get corresponding duration */

  printf(" %d        ", indices[0]);
  for(t=1; t < nt; t++)
   printf("%s", (t >= XPRMgetvsol(mod,s) && t < XPRMgetvsol(mod,s)+D) ?
        "*" : " ");
  printf("  (Start/Duration %g/%d)\n", XPRMgetvsol(mod,s), D);
 } while (!XPRMgetnextarrentry(varr, indices));

 return 0;
}
```

Note that we have been able to get model parameters (NJ, NT), and data table values (table D). We have retrieved optimal values into the program. Here is the same application written in Java.

```java
import com.dashoptimization.*;

public class runsched
{
 public static void main(String[] args) throws Exception
 {
  XPRM mosel;
  XPRMModel mod;
  XPRMArray varr, darr;
  XPRMMPVar s;
  int[] indices;
```

```
        int nt, t, D;

        mosel = new XPRM();                 // Initialize Mosel

        mod = mosel.loadModel("sched.bim"); // Load a bim file
        mod.run();                          // Run & optimize the model

        if(mod.getProblemStatus()!=mod.PB_OPTIMAL)
         System.exit(1);                    // Stop if no solution found

        System.out.println("Minimum makespan " + mod.getObjectiveValue());

        varr=(XPRMArray)mod.findIdentifier("s");  // Get the model object 's'
        darr=(XPRMArray)mod.findIdentifier("D");  // Get the model object 'D'
        nt=((XPRMConstant)mod.findIdentifier("NT")).asInteger();
                                            // Get the model object 'NT'

// Print a little bar chart
        System.out.println("Job    Time:1234567890");
        indices = varr.getFirstIndex();     // Get 1st entry of array varr
        do
        {
         s = varr.get(indices).asMPVar();   // Get a variable from varr
         D = darr.getAsInteger(indices);    // Get corresponding duration

         System.out.print(" " + indices[0] + "          ");
         for(t=1; t < nt; t++)
          System.out.print((t >= s.getSolution() && t < s.getSolution()+D) ?
              "*" : " ");
         System.out.println("  (Start/Duration " + s.getSolution() + "/" + D + ")");
        } while(varr.nextIndex(indices));   // Get the next index

     }
    }
```

5.3   BCL from C

Here is the same model, written in C using BCL.

```
        #include <stdio.h>
        #include "xprb.h"

        #define NJ  4               /* Number of jobs */
        #define NT 10               /* Time limit */

        int D[NJ] = {3, 4, 2, 2};   /* Durations of jobs   */

        XPRBvar s[NJ];              /* Start times of jobs  */
        XPRBvar delta[NJ][NT];      /* Binaries for start times */
        XPRBvar z;                  /* Maximum completion time (makespan) */
        XPRBprob p;                 /* A problem */

        void model(void);           /* The BCL model */
        void solve(void);           /* Solving and solution printing */

        int main(int argc, char **argv)
        {
          model();                  /* Formulation */
          solve();                  /* Solve and print solution */
          return 0;
        }


        void model(void)            /* BCL formulation */
        {
         XPRBctr ctr;
```

```
     int j, t;

     p = XPRBnewprob("Jobs");    /* Initialize BCL & create a new problem */

/**** Create variables ****/
 for(j = 0; j < NJ; j++)
  s[j] = XPRBnewvar(p, XPRB_PL, XPRBnewname("s_%d",j+1), 0, NT-D[j]+1);

 z = XPRBnewvar(p, XPRB_PL, "z", 0, NT);

 for(j = 0; j < NJ; j++)
  for(t = 0; t < NT-D[j]+1; t++)
   delta[j][t]=XPRBnewvar(p, XPRB_BV, XPRBnewname("delta_%d%d",j+1,t+1),
                          0, 1);

/**** Constraints ****/
    /* Calculate maximum completion time of all jobs */
 for(j = 0; j < NJ; j++)
 {
  ctr = XPRBnewctr(p, XPRBnewname("C1_%d",j), XPRB_G);
  XPRBaddterm(ctr, z, 1);
  XPRBaddterm(ctr, s[j], -1);
  XPRBaddterm(ctr, NULL, D[j]);
 }

    /* Precedence relations between two pairs of jobs */
    /* C2_31:   3 must follow 1 */
 ctr = XPRBnewctr(p, "C2_31", XPRB_G);
 XPRBaddterm(ctr, s[2], 1);
 XPRBaddterm(ctr, s[0], -1);
 XPRBaddterm(ctr, NULL, D[0]);

    /* C2_41:   4 must follow 1 */
 ctr = XPRBnewctr(p, "C2_41", XPRB_G);
 XPRBaddterm(ctr, s[3], 1);
 XPRBaddterm(ctr, s[0], -1);
 XPRBaddterm(ctr, NULL, D[0]);

    /* Relation linking start time of jobs with corresponding binary */
 for(j = 0; j < NJ; j++)
 {
  ctr = XPRBnewctr(p, XPRBnewname("C3_%d", j+1), XPRB_E);
  for(t = 0; t < NT-D[j]+1; t++) XPRBaddterm(ctr, delta[j][t], t+1);
  XPRBaddterm(ctr, s[j], -1);
 }

    /* One start time for each job */
 for(j = 0; j < NJ; j++)
 {
  ctr = XPRBnewctr(p, XPRBnewname("C4_%d",j+1), XPRB_E);
  for(t = 0; t < NT-D[j]+1; t++) XPRBaddterm(ctr, delta[j][t], 1);
  XPRBaddterm(ctr, NULL, 1);
 }

/**** Objective ****/
 ctr = XPRBnewctr(p, "MINIM", XPRB_N);
 XPRBaddterm(ctr, z, 1);
 XPRBsetobj(p, ctr);               /* Select objective function */
}


void solve(void)
{
 int statmip, j;

 XPRBsetsense(p, XPRB_MINIM);
 XPRBsolve(p, "g");                /* Solve the problem as MIP */
```

```
statmip = XPRBgetmipstat(p);    /* Get the MIP problem status */

if((statmip == XPRB_MIP_SOLUTION) || (statmip == XPRB_MIP_OPTIMAL))
{                               /* An integer solution has been found */
 printf(" Min makespan is %g\n", XPRBgetobjval(p));
 for(j = 0; j < NJ; j++)        /* Print solution for all start times */
  printf(" %s starts at time %g\n", XPRBgetvarname(s[j]),
        XPRBgetsol(s[j]));
}
}
```

5.4   BCL from C++

```cpp
#include <iostream>
#include "xprb_cpp.h"

using namespace std;
using namespace ::dashoptimization;

#define NJ  4                   // Number of jobs
#define NT 10                   // Time limit

/**** DATA ****/
double D[] = {3,4,2,2};         // Durations of jobs

XPRBvar s[NJ];                  // Start times of jobs
XPRBvar delta[NJ][NT];          // Binaries for start times
XPRBvar z;                      // Maximum completion time (makespan)
XPRBsos set[NJ];                // Sets regrouping start times for jobs

void model(void);               // Basic model formulation
void solve(void);               // Solving and solution printing

XPRBprob p("Jobs");             // Initialize BCL and a new problem

int main(int argc, char **argv)
{
  model();                      // Problem definition
  solve();                      // Solve and print solution
  return 0;
}

void model()
{
 XPRBexpr le;
 int j, t;

                                // Create start time variables
 for(j=0; j<NJ; j++) s[j] = p.newVar("start", XPRB_PL);

 z = p.newVar("z",XPRB_PL,0,NT);  // Declare the makespan variable

 for(j=0; j<NJ; j++)            // Declare binaries for each job
  for(t=0; t<(NT-D[j]+1); t++)
   delta[j][t] = p.newVar(xbnewname("delta_%d%d",j+1,t+1), XPRB_BV);

/**** Constraints ****/
 for(j=0; j<NJ; j++)            // Calculate maximal completion time
  p.newCtr("C1", s[j]+D[j] <= z);

 p.newCtr("C2_31", s[0]+D[0] <= s[2]);  // 3 must follow 1
 p.newCtr("C2_41", s[0]+D[0] <= s[3]);  // 4 must follow 1

 for(j=0; j<NJ; j++)            // Linking start times and binaries
```

```
    {
     le = 0;
     for(t=0; t<(NT-D[j]+1); t++)     le += (t+1)*delta[j][t];
     p.newCtr(xbnewname("C3_%d",j+1), le == s[j]);
    }

    for(j=0; j<NJ; j++)              // One start time for each job
    {
     le = 0;
     for(t=0; t<(NT-D[j]+1); t++)     le += delta[j][t];
     p.newCtr(xbnewname("C4_%d",j+1), le == 1);
    }

/**** Objective ****/
    p.setObj(p.newCtr("OBJ", z));    // Define and set objective function


    for(j=0; j<NJ; j++) s[j].setUB(NT-D[j]+1);
                                     // Upper bnds on start time variables

/**** Output ****/
    p.print();                       // Print out the problem definition
   }

   void solve()
   {
    int statmip, j;

    p.setSense(XPRB_MINIM);          // Say we are minimizing
    p.solve("g");                    // Solve the problem as a MIP
    statmip = p.getMIPStat();        // Get the MIP problem status

    if((statmip == XPRB_MIP_SOLUTION) || (statmip == XPRB_MIP_OPTIMAL))
    {                                // An integer solution has been found
     cout << " Min makespan is " << p.getObjVal() << endl;
     for(j=0; j<NJ; j++)             // Print solution for all start times
      cout << s[j].getName() << ": " << s[j].getSol() << endl;
    }
   }
```

5.5 BCL from Java

Here it is in Java.

```
import com.dashoptimization.*;

public class schedjava
{
 static final int NJ = 4;        /* Number of jobs */
 static final int NT = 10;       /* Time limit */

 static final double[] D = {3,4,2,2};   /* Durations of jobs */

 static XPRB bcl;
 static XPRBvar[] s;             /* Start times of jobs */
 static XPRBvar[][] delta;       /* Binaries for start times */
 static XPRBvar z;               /* Maximum completion time (makespan) */
 static XPRBprob p;              /* A problem */

 static void model()
 {
  XPRBexpr le;
  int j, t;

  s = new XPRBvar[NJ];           /* Create start time variables */
  for(j=0; j<NJ; j++) s[j] = p.newVar("start", XPRB.PL);
```

```
      z = p.newVar("z",XPRB.PL,0,NT);   /* Declare the makespan variable */

   delta = new XPRBvar[NJ][NT];
   for(j=0; j<NJ; j++)             /* Declare binaries for each job  */
    for(t=0; t<(NT-D[j]+1); t++)
     delta[j][t] = p.newVar("delta"+(j+1)+(t+1), XPRB.BV);

   for(j=0; j<NJ; j++)             /* Calculate maximal completion time */
    p.newCtr("C1", s[j].add(D[j]).lEql(z) );

                                   /* Prec. rel. betw. 2 pairs of jobs */
   p.newCtr("C2_31", s[0].add(D[0]).lEql(s[2]) );
   p.newCtr("C2_41", s[0].add(D[0]).lEql(s[3]) );

   for(j=0; j<NJ; j++)             /* Linking start times and binaries  */
   {
    le = new XPRBexpr();
    for(t=0; t<(NT-D[j]+1); t++)  le.add(delta[j][t].mul((t+1)));
    p.newCtr("C3_"+(j+1),         le.eql(s[j]) );
   }

   for(j=0; j<NJ; j++)             /* One start time for each job  */
   {
    le = new XPRBexpr();
    for(t=0; t<(NT-D[j]+1); t++)  le.add(delta[j][t]);
    p.newCtr("C4_"+(j+1),         le.eql(1));
   }

   p.setObj(z);                    /* Define and set objective function */

   for(j=0; j<NJ; j++) s[j].setUB(NT-D[j]+1);
                                   /* Upper bnds on start time variables */

   p.print();                      /* Print out the problem definition */
  }


 static void solve()
 {
  int statmip, j, t;

  p.setSense(XPRB.MINIM);
  p.solve("g");                    /* Solve the problem as MIP */
  statmip = p.getMIPStat();        /* Get the MIP problem status */

  if((statmip == XPRB.MIP_SOLUTION) || (statmip == XPRB.MIP_OPTIMAL))
  {                                /* An integer solution has been found */
   System.out.println("Objective: "+ p.getObjVal());
   for(j=0;j<NJ;j++)               /* Print solution for all start times */
    System.out.println(s[j].getName() + ": "+ s[j].getSol());
  }
 }


 public static void main(String[] args)
 {
  bcl = new XPRB();                /* Initialize BCL */
  p = bcl.newProb("Jobs");         /* Create a new problem */
  model();                         /* Problem definition */
  solve();                         /* Solve and print solution */
 }

}
```