

Xpress-MP でモデルを構築する

サマリー

このペーパーは、Xpress-MP を使ってアプリケーション開発者が、モデルを構築し、解くことに使えるオプションについて説明したものです。

**[皆様の利便のための参考翻訳です。誤訳があるかもしれませんので、
原文をお読みなるように、強く、お勧めいたします。]**

イントロダクション

モデルの構築は、多分、数理計画法を実用的な目的で使用する時、最も知的でやりがいがある部分でしょう。しかし、ひとたび、モデル化しようとする状況を、数式的なモデルで表現し終わると、次は、何らかのコンピュータシステム環境の中でモデルを実行する方法を決めなければなりません。このとき、考慮しなければならないのは、通常、下記のようなことです。

- モデルが正しいことを確認する
- モデルのメンテナンス、および、修正が容易であること
- 解くために使用するアルゴリズム
- データへのアクセスとハンドリング
- モデルを解くスピード
- 市場に提供するのに必要なスピード

このペーパーでは、XpressMP を使用するとき、アプリケーション開発者である皆様に、モデルを構築し、解く場合に開かれているオプションについて議論します。このペーパーでは、特に、下記の 3 つのアプローチを使用するとき、それらのアプローチの持つ長所と短所を、相対的に評価します。

- モーゼルを使ってモデルを構築するときの、モデル構築、および、最適化の環境、ならびに、その言語、そして、Mosel ライブラリと Xpress-Optimizer ライブラリを利用して、これらをどのように使う。
- 皆さんのアプリケーションの中で、Xpress-Optimizer ライブラリとともに、Xpress Builder Library BCL を使って、モデルを構築し、配置する。
- 皆さんのアプリケーションの native language を使ってモデルを構築し、配置して、問題事例を、直接、Xpress-Optimizer ライブラリにロードする。

アベンディックスで、小さいモデルを取り上げ、それを Mosel のモデル作成言語で表現し、BCL(C、C++、および、Java) を使う例を示しています。

Mosel 言語

Mosel のモデル作成言語は、キーボードで利用可能な文字により制約されていますが、できるだけ、モデルの数式的な定式化に近い形式で表現できるようにデザインされています。次の例を見てください。

$$\sum_{t=1}^{NT-D_j+1} t\delta_{j,t} = s_j, \quad j = 1, \dots, NJ$$

上の式は、Mosel 言語では、下記のように表現されます。

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

複雑なモデルでも、理解し易い、そして、迅速に実行できる簡単なシンタックスを使って、容易に構築できます。下記は、そのようなモデルの一部です。

```
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 ...
  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
```

この他にも、index set、partial integers、semi-continuous variables、special ordered sets と model cuts、そして、integer programming directive などの、強力な整数計画法のモデル作成機能を持っています。さらに、looping、selection、ranges と sets、その他のモデル構築に必要な機能を持つ、非常に強力なプログラミング言語があります。

ソフトウェアツール

Dash 社は、モデルを開発して、維持するための、一連の高性能のツールを提供しております。モデルを開発し、デバッグするための最良のツールは Xpress-IVE で、これは、Windows で使用できる統合モデル作成と最適化のための開発環境です。

Mosel は、すべてのコンピュータプラットフォームで行うモデルの開発に適したモデル作成と最適化を行うための環境です。モデルは、VB、C/C++、Java、または、C#などのプログラミング言語から

「コール」でき、また、Mosel ライブラリを使って、アプリケーションの中に埋め込むことができます。したがって、Mosel は、異なった、使用可能なソフトウェアに対応し、それぞれ別のパーツ/インタフェースを持っています。すなわち、スタンドアロンでの使用には「コマンドライン」、C/C++などに書かれた既存のアルゴリズムの統合と使用を可能にするライブラリ、それに Xpress-IVE の基礎をなす言語です。

明瞭性と使い易さ

モデルの表現は数式に近いので、通常、モデルは、非常に短くて、理解しやすくなっています。このため、モデル作成言語は、短時間で容易に習得でき、実際にすぐに適用できるので、モデルを構築し、理解し、変更するのはたやすいことです。モデルは、簡単なテキストファイルから読み込まれます。その修正は容易で、アプリケーションの中に統合することが出来ますが、それには、モデルを再構築する必要はありません。Mosel は、モデルをコンパイルできます。そして、コンパイルされたモデルは、どんなプラットフォームでも実行可能です。

セキュリティ

Mosel を使うと、アプリケーションを構築し、理解し、変更し、維持するのが簡単になります。多くのアプリケーションで、開発者は、エンドユーザが、モデルファイルを読み、変更するのを防止したいと思っています。例えば、知的所有権に関連した商業上の理由とか、エンドユーザが手を加えると、品質や信頼性に悪影響を与えるかもしれないという懸念があるでしょう。Mosel でコンパイルされた (BIM) ファイルは、モデルをエンドユーザの目から、完全に隠してくれます。

データアクセスとハンドリング

Mosel の中でデータを操作する方法は、ハイレベルのプログラミング言語に較べ、それらに勝ることはないかも知れませんが、それと同じくらい強力です。Mosel は、非常に高水準のデータ・インターフェース機能を提供しています、そこでは、モデルの中の一行で、データのインポート、エクスポートが出来ます。

Mosel は、Mosel 自身の形式のファイルを含み、いくつかの異なるデータへのアクセス方法を提供していますが、それらは、非常にセットアップしやすいものです。すなわち、フリーフォーマットのテキストファイル、メモリに保持されているデータ、アプリケーションで生成されたデータです。さらに、(追加的な)ODBC を経由して、ODBC インタフェースを持っているどんなデータベースへもアクセスでき、Microsoft Excel スプレッドシートへのアクセスも可能です。そして、ユーザは、自由に、新しいデータソース、フォーマットを定義できます。

最適化

Mosel は、線形、整数、二次計画法問題を解くのに、Xpress-Optimizer ライブラリを使います。拡

大モジュールにより、Xpress suite の他のソルバーへのアクセスが出来ます。Xpress suite の他のソルバーには、非線形計画問題を解くための Xpress-SLP、確率モデルを解くための Xpress-SP、そして、有限ドメインと浮動小数点制約ソルバーを使用するための Xpress-Kalis があります。また、Mosel は、特別なソルビング・テクニックのための、例えば、Tabu Search のようなソルバーヘインターフェイスすることも出来ます。

サマリー

モデルが正しいことの確認	容易
メンテナンスと変更	容易
アルゴリズム	内部的
データアクセスとハンドリング	容易で、かつ、高いレベル
モデル実行スピード	Mosel、XOSL に較べ、ごく僅か、低いこともある
市場への提供スピード	早い
なぜ、使うのか？	アプリケーションのうち、その 95%が市場に早く提供できる

BCL

BCL の背後にある考え方は、C、C++、Java、C#、VB のようなハイレベルのプログラミング言語の中で、マトリクスを作るのにライブラリを使おうということです。マトリクスは、少しずつ、前もって定められた順序なしに、非常にフレキシブルに構築できます。ここで、C 言語を使って、方程式が、どのように書かれるかを見てみましょう。まず、変数を、すべて、左辺に移項します。

$$\sum_{t=1}^{NT-D_j+1} t\delta_{j,t} - s_j = 0, \quad j = 1, \dots, NJ$$

そうすると、コードは下記のようなになるでしょう。ここで、BCL ファンクションは、太字で示されています。

```
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);
}
```

ここでは、C の宣言 (declaration) は、省略してあります。問題 prob のコンテキストでは、XPRBnewctr() のコールにより、新しい制約式 (厳密に言えば、新しい制約式へのポインタ) が生成され、XPRB_E により、それが等式であるということがしていされます。この時点では、この制約式は空っぽですが、XPRBaddterm() により、値 $t+1$ を持つ項が、変数 $\text{delta}[j][t]$ の制約式に加えられます。ここで、これを行うために、C 言語のループ機能を使い、対象となる t の値について、可能な j の値のすべてに関してループさせます。最後に、もう一度、XPRBaddterm() を使い、 -1 の係数を持つ制約式に、変数 $s[j]$ を加えます。デフォルトで、制約式の右辺は 0 です。

セキュリティ

BCL は、1 セットのルーチンのコールにより行われますので、モデルは、ハイレベルのプログラムの中に完全に隠されています。それをコンパイルすると、エンドユーザはモデルにアクセスできません。

データアクセスとハンドリング

プログラミング言語により、すべてのデータハンドリング機能が提供されます。ODBC では、これは簡単ではないかもしれませんが、プログラムを作る用意ができている限り、どんなソースにもアクセスできます。さらに、取得したデータに関して、どんな操作や変換も行えます。また、確認テストを行ったり、複雑な診断なども書いたりすることもできます。

モデルのフレキシビリティ

一般に、既にモデルがプログラミング言語でコード化されてしまっているとき、それらに柔軟性を持たせることは困難です。しかし、例えば、読者の皆さんが、Mosel モデルを構築し、デバッグして、エンドユーザに配布してあると仮定してください。そこで、そのモデルに、例えば、一組の制約式を加え、モデルを変更したいような状況が起こったとします。このような場合、読者の皆さんが行なわなければならないのは、変更が加えられたモデルファイルをメールすることです。しかし、もし、プログラミングインターフェースを使っているとすると、アプリケーションを再構築しなければならず、それには、たくさんの時間をテストするのに費やして、そして、かなり大きい実行ファイルを送らなければなりません。

サマリー

モデルが正しいことの確認	非常に容易 (Mosel よりも難しく、XOSL よりも容易)
メンテナンスと変更	Mosel よりも難しく、XOSL よりも容易
アルゴリズム	モデル構築が容易なアルゴリズムで、モデルの構造を利用する
データアクセスとハンドリング	ネイティブプログラミング言語、ハイレベルの BCL、固有
モデル実行スピード	おそらく、Mosel よりも速く、XOSL と同等

市場への提供スピード	XOSL より速く、Mosel より遅い
なぜ、使うのか？	アプリケーションの中での、モデル構築、および、変更が、 完全な柔軟性を持って行える

オプティマイザーライブラリ

Xpress-Optimizerライブラリ・インタフェースは、オプティマイザの最も低いレベルです。プログラミング言語の中で、オプティマイザが必要とするデータ構造をセットアップし、それをライブラリに渡します。次いで、ライブラリは、いろいろなことをするよう命令されます。例えば、まず、モデルを解く、続いて、得られた最適値に、プログラムからアクセスできます。

インタフェースは、必要があって、結構、複雑ですが、整数計画問題をライブラリにロードするルーチンのスペックを見ると、インタフェースに必要な事項について、それを理解する手がかりが得られます。そこには、下記のような、23 ものアーギュメントがあります！

```
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);
```

ここで、下記のことを指定してやらなければなりません。

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

1つの大きな難題は、制約マトリクスの最初のコラムのノンゼロ係数を指定し、続いて、2番目、3番目と、順次、ノンゼロ係数を指定しなければならないことです。したがって、定式化されたモデルから制約式を取ったり、追加したりすると、このプログラムに、大きな変更を加えることが必要になります。

BCL でモデルを構築することに比べ、ライブラリで「モデルを構築すること」の、唯一の本当の利点は、ライブラリの方が、非常にわずかだけ速いということだけです。なぜならば、BCLの場合、Optimizer ライブラリヘデータを渡す前に、BCL 自身が、loadglobal() format にデータ変換しなければならないからです。これには、ほとんど時間がかかりませんが、直接、Optimizer ライブラリに連結して、プログラムでこれを行うと、明らかに、わずかですが、このオーバーヘッドが回避できます。(もちろん、BCLで生成されたマトリクスにも、Optimizerライブラリの非常に強力なフィーチャーを使えるので、BCL でモデルを構築することで失われるものは、この、ほんの少しのスピード以外、何もありません。)

2つの理由から、ここでは、直接、Optimizer ライブラリ・インタフェース向けに書かれた式を示しません。1番目の理由は、インタフェースがアクティビティ (コラム)オリエンテッドであること、2番目の理由は、それが悪夢であるということからです。

これまで、伝統的に、Optimizerライブラリのような最適化サブルーチンライブラリは、オブティマイゼーションを使う、効率的なアプリケーションを書く唯一の手段でした。そして、そのようなライブラリは、このために広く使用されています。しかし、Mosel言語とBCLは、問題を構築し、問題を操作するという仕事を容易にできるので、ユーザに本当の便宜を提供することができ、しかも、アプリケーションのスピードにも、ほとんど負担を掛けません。

サマリー

ダイレクト・オブティマイザー・インターフェースの利点/難点 について、いろいろなことが言われていますが、それらは、下記のようなものでしょう。そして、それらは、多分、偏った見方でしょう。

モデルが正しいことの確認	非常に困難
メンテナンスと変更	非常に難しい
アルゴリズム	モデルを構築しやすいアルゴリズム
データアクセスとハンドリング	ネイティブプログラミング言語
モデル実行スピード	多分、最も速い
市場への提供スピード	最も遅い
なぜ、使うのか？	最も効率的 - しかし、メンテナンスと変更の容易さを失う

A Complete Model

Mosel

下記は、Mosel で書かれたあるモデルの全体を示したものです。

```

model sched
  uses "mxxprs"                                ! 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

```

Mosel Runtime

Mosel ランタイムでは、同じモデルを使用できます。この例は、C 言語で書かれた典型的なアプリケーションで、コンパイルしたモデルをロードして、MIP を解き、小さい棒グラフをプリントします。ここで、コンパイルされたモデルは、ファイル sched.bim にあるとしています。

```

#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;
}

```

モデルパラメタ(NJ, NT)、および、データテーブルの値 (table D) が得られたことに注意してください。そして、プログラムに最適値をリトリブしました。

以下は、同じアプリケーションを Java で書いたものです。

```

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

}
}

```

BCL from C

以下は、同じモデルを、BCL を使って、C 言語で書いたものです。

```

#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]));
}
}

```

BCL from C++

```

#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( )
{
    XPRBlinExp 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))
    {

```

```

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;
}
}

```

BCL from Java

以下は、Java で掛かれたものです。

```

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 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( )
    {
        XPRBlinExp 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]).IEql(z) );

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

    for(j=0; j<NJ; j++)                                /* Linking start times and binaries */
    {
        le = new XPRBlinExp( );
        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 XPRBlinExp( );
        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 */
}
}
```