Moselを使って、複数のモデルを管理し、パラレルに解く*

Dash Optimization Whitepaper

Last update 18 October, 2006

Multiple models and parallel solving with Mosel

Y. Colombani and S. Heipcke

Dash Optimization, Blisworth House, Blisworth, Northants NN7 3BX, UK

Yves.Colombani@dashoptimization.com,

Susanne.Heipcke@dashoptimization.com

18 October, 2006

アブストラクト

この論文では、Moselを使い、複数のモデルを、順次、そして、並行的に解く方法を、いくつかの例で説明します。可能なコンフィギュレーションのすべてを網羅的にリストすることができないので、この小論では、例を使って、Moselモジュール*mmjobs*の使い方を紹介します。例としては、あるモデルのいくつかのインスタンスをコンカレントに実行する、マスターに、順次、サブモデルを埋め込む、ディコンポジション・アルゴリズム(Dantzig-Wolfe and Benders decomposition)を実行するなどを使います。また、より専門的な観点から、小論では、モデル管理、コンカレント・モデルのシンクロナイゼーション、共有メモリIOドライバーの使い方なども議論します。

* この翻訳は、あくまでも、皆様のご参考に供するためのものですので、内容的の厳密性は、保障できません。したがって、英文のWhitepaperもお読みなさることをお勧めいたします。英文は、下記より入手できます。
http://jp.dashoptimization.com/home/services/publications/support_papers.html

目次

1 イントロダクション

モーゼルのリリース1.6では、モーゼル言語を使い、直接、複数のモデルを扱うことができるようになりました。モジュール*mmjobs*によって、新たに提供されるようになった機能には、*モデル管理機能*、イベントの待ち行列に基づく*コンカレント・モデルのシンクロナイゼーション*機能、および、*共有メモリIOドライバー*が含まれています。以下のリストは、利用可能な機能の概要を記したものです。このモジュールについての詳細は、Mosel Reference Manualの*mmjobs*の部分をご参照ください。

- **モデル管理:** モデルのソースファイルのコンパイル、bim ファイルのロード、モデルの実行と中断、モデル情報(status, exit code, ID)のリトリーバル、IO ストリームの出力先変更。
- **シンクロナイゼーション・メカニズム:** イベントの送信、リトリーブ、イベント、もしくは、イベントクラスの待ち、イベント情報(class, value, sender model)のリトリーブ
- **共有メモリIOドライバー:** コンカレント・モデル間でデータをやりとりするための、memドライバーの共有メモリ・バージョン(write access by a single model, read access by several models simultaneously)、これは、Moselがファイルネームを予期している場所ならどこでも、とくに、initializations blockで使用できる。
- **メモリ・パイプIOドライバー:** コンカレント・モデル間でデータをやりとりするためのメモリ・パイプ(write access by several models, read access by a single model)で、これは、Moselがファイルネームを予期している場所ならどこでも、とくに、initializations blockで使用できる。

*mmjobs*により、新たに二つの「タイプ」が導入されました。すなわち、Modelと Eventです。Modelは、Moselモデルの参照に使われます。モデルの参照を行う前に、bimファイルをロードしてイニシアライズしておかなければなりません。

Eventは、Mosel言語の中で、一つのeventを意味します。Eventは、*class*と*value*で特徴づけられ、モデルと*parent*モデルの間でやりとりできます。各モデルには、event queueが取り付けられ、このモデルに送られてくるすべてのeventを受け取ります。その管理は、FIFO (First In – First Out)で行なわれます。

この論文の最初の章では、読者に、Moselで、下記のようなモデルを扱うときに、通常行なわれる基本的なタスクを説明します。

- マスターモデルからサブモデルを実行する：その順序は、コンパイル、ロード、ラン、ウエイトの順です。
- サブモデルの実行を止める
- サブモデルからアウトプットを得る： (特に、IVEの下では)サブモデルを静かにする

ため、ファイルに出力する
- メモリへコンパイルする
- ランタイムパラメータを渡す
- 複数のサブモデルのランを行なう
  - 順繰りに
  - 同時並行的に
- モデル間でデータをやり取りする：共有メモリIOドライバー、メモリパイプIOドライバーを使う

この論文の残りの部分で、*mmjobs*を使って行なう、もう少し進んだ例題について議論します。そこでは、それらのインプリメンテーションの仕方の詳細も説明します。これらの例題は、すべて、Dash websiteからダウンロードできます。

- カラムの生成： Moselユーザガイドに掲載されている二つの別個のカラム生成のモデルを、共有メモリ経由でデータを渡し順番に解く。
- パラレルで解く： 同一のモデルのいくつかのインスタンスを、別個の解法アルゴリズムパラメータで、コンカレントにランする。改善された解の値は、ランが行なわれているすべてのモデルに送られ、バウンドを更新する。そして、最も早く最適解を得たモデルは、他のモデルをすべて停止させる。
- Dantzig-Wolfeディコンポジション： コンカレントな方法でサブ問題の一組のインスタンス（subproblem instances）を解くときの反復手順、更新マスター問題を解く。データのやりとりは、共有メモリ経由で行なう。
- Benders ディコンポジション： それぞれに異なる一連のサブ問題を繰り返して解く。データのやりとりは、共有メモリ経由で行なう。

ここで強調しておきたいのは、*multiple models*と*multiple problems*の違いです。 — Moselは、複数のモデルを扱えますが、これらの個々のモデルは、皆、単一の問題に対応したものです。これは、例えば、ある一つのモデルが、Xpress-Optimizerのようなソルバーを、何回かcallするような場合、 ソルバーは、単一の問題を表現したものを扱い、常に、最後に行なった最適化ランのソリューションのみが得られることを意味します。

2基本的なタスク

この章では、Mosel でいくつかのモデルを扱うときに、*mmjobs*モジュールの機能を使って行なう基本的なタスクについて説明します。

2.1 サブモデルを実行する

ここで、下記の簡単なtestsub.mosというサブモデルを、その*master model*から実行したいとします。

```
model "Test submodel"

 forall(i in 10..20) write(i^2, " ")
 writeln

end-model
```

ここで、読者の皆さんは、標準的なcompile-load-runという手順に慣れていると前提します。この手順は、Mosel command line、ホストのアプリケーション、別のMoselモデルなど、どこから実行するに関りなく、Moselモデルの実行に、常に、必要な手順です。Moselモデルを*master model*から実行する場合、compile-load-run-wait という標準的な手順を補強する必要があります。これの背後にある理論的説明は、サブモデルを別の一つのスレッドとして開始し、こうすることで、そのサブモデルが、*master model* が終わる前に終了する（もしくは、*master model*が、サブモデルの結果を利用して、実行を継続する－後述の2.7節を参照）ようにさせるためです。

下のモデルruntestsub.mosは、上のtestsub.mosというモデルを実行するため、compile-load-run-wait という手順を、その基本的な形式で使います。それに対応するMosel のサブルーチンは、*mmjobs*で定義しますが、これは、usesステートメントの中に含まれている必要があります。サブモデルはそのままです（すなわち、サブモデル自体がこのモジュールの持つ機能を何も使わない限り、サブに*mmjobs*を入れる必要はありません）。master modelの最後のステートメントdropnexteventについては、もう少し説明が必要でしょう。サブモデルの終了は、master modelに送られるevent of class EVENT_ENDで知らされます。waitステートメントは、master modelの実行を、eventを受け取るまで、一時、休止させます。ここでの例の場合、サブモデルが送る唯一のevent がこの終了メッセージなので、その種類を確認することなく、master modelのevent queue から、メッセージを削除します。

```
model "Run model testsub"
 uses "mmjobs"

 declarations
  modSub: Model
end-declarations

                                        ! Compile the model file
```

```
  if compile("testsub.mos")<>0 then exit(1); end-if
  load(modSub, "testsub.bim")                    ! Load the bim file
  run(modSub)                                    ! Start model execution
  wait                                           ! Wait for model termination
  dropnextevent                                  ! Ignore termination event message

end-model
```

この二つのモデルは、Mosel modelの標準中的な実行方法(Mosel command line, within Xpress-IVE, or from a host application)のいずれからも、master model を実行することでランされます。Mosel command line から実行する場合、例えば､下のようなコマンドを使います。

```
mosel -c "exec runtestsub.mos"
```

その結果として、サブモデルにより、下のようなプリントが出てきます（Xpress-IVEを使っている読者は、2.3節を参照してください）。

```
100   121   144   169   196   225   256   289   324   361   400
```

サブモデルの終了ステータスについて、より正確な情報を得たいときは、dropnexteventというステートメントを、下記のステートメントに置き換えてください。これにより、サブモデルにより送られるevent がリトリーブされ、そのクラス(the termination event has the predefined class EVENT_END) 、および、そのバリュー(default value 0)がプリントされます。それに加え、モデルのexit code が表示されます(value sent by an exit statement terminating the model execution or the default value 0)。

```
declarations
  ev: Event
end-declarations

ev:=getnextevent
writeln("Event class: ", getclass(ev))
writeln("Event value: ", getvalue(ev))
writeln("Exit code : ", getexitcode(modSub))
```

2.2 サブモデルの実行を停止する

仮に、サブモデルの実行に長い時間が掛かるような場合、master modelを停止することなく、サブモデルを停止したほうが望ましいことがあります。master modelを停止することなく、サブモデルを停止するには、下記のようにmaster model(file runsubwait.mos)を修正し、waitステートメントにduration (in seconds)を加えてください。 サブモデルが、1秒、実行した後、termination event message を送っていない場合、サブモデルは、stop のcall により停止されます。このとき、model reference が表示されます。

```
model "Run model testsub"
 uses "mmjobs"

 declarations
  modSub: Model
 end-declarations

                                              ! Compile the model file
 if compile("testsub.mos")<>0 then exit(1); end-if
 load(modSub, "testsub.bim")                  ! Load the bim file
 run(modSub)                                  ! Start model execution
 wait(1)                                      ! Wait 1 second for an event

 if isqueueempty then                         ! No event has been sent: model still runs
  writeln("Stopping the submodel")
  stop(modSub)                                ! Stop the model
  wait                                        ! Wait for model termination
 end-if
 dropnextevent                                ! Ignore termination event message

end-model
```

2.3 サブモデルのアウトプット

これまでの説明に基づき、Xpress-IVEでmaster modelを実行した読者の皆さんは、サブモデルからのアウトプットを何もご覧になれなかったと思います。これは、IVE がmaster model のアウトプットしか捉えないからです。この場合、サブモデルのアウトプットにアクセスする一番よい方法は、このアウトプットの出力先をファイルに変更することです。サブモデルのアウトプットの出力先を変更することは、いくつかの（サブ）モデルを同時並行的にランしている場合にも必要です。なぜなら、スクリーン上に表示されるアウトプットは、これらのモデルからのアウトプットが混ざって表示されるからです。

下記のようなステートメントを使い、サブモデルの中で、アウトプットがプリントされる前に、直接、アウトプットの出力先を変更するのがよいでしょう。

```
fopen("testout.txt", F_OUTPUT+F_APPEND)      ! Output to file (in append mode)
fopen("tee:testout.txt&", F_OUTPUT)          ! Output to file and on screen
fopen("null:", F_OUTPUT)                     ! Disable all output
```

ここで、最初のステートメントは、アウトプットをtestout.txtというファイルに出力させ、2行目のステートメントは、testout.txtというファイルに書き込み中、アウトプットをスクリーンに保持し、そして3番目のステートメントは、モデルを完全にsilentにします。アウトプット・ファイルは、モデルで、プリント・ステートメントの後に、fclose(F_-OUTPUT)というステートメントを追加することによりクローズされます。

同様のことが、master modelからも行なえます。その方法は、対応するサブモデルのrun ステートメント前に、下記のように、アウトプットの出力先変更を加えます。

```
setdefstream(modSub, F_OUTPUT, "testout.txt")    ! Output to file
setdefstream(modSub, F_OUTPUT, "tee:testout.txt&")
                                                 ! Output to file and on screen
```

```
setdefstream(modSub, F_OUTPUT, "null:")          ! Disable all output
```

サブモデルでのアウトプットの出力先変更は、アウトプット・ストリームをリセットすることで終了できます。

```
setdefstream(modSub, F_OUTPUT, "")
```

2.4 メモリへのコンパイル

Moselファイル*filename*.mosのディフォルトのコンパイルにより、*filename*.bimというバイナリーのモデル・ファイルが生成されます。サブモデルの物理的なBIM fileの生成を避けるため、下の例runsubmem.mosに示されているように、サブモデルをメモリにコンパイルすることもできます。物理的なBIM fileにアクセスするよりも、メモリを使うと、通常、効率性がよくなります。さらに、master modelが実行されている場所にwrite accessができない場合、この機能は便利です。

```
model "Run model testsub"
 uses "mmjobs", "mmsystem"

 declarations
  modSub: Model
 end-declarations
                                                  ! Compile the model file
 if compile("", "testsub.mos", "shmem:testsubbim")<>0 then
  exit(1)
 end-if
 load(modSub, "shmem:testsubbim")                 ! Load the bim file from memory
 fdelete("shmem:testsubbim")                      ! ... and release the memory block
 run(modSub)                                      ! Start model execution
 wait                                             ! Wait for model termination
 dropnextevent                                    ! Ignore termination event message

end-model
```

compileは3つのアーギュメントが必要です。すなわち、compilation flags (例えば、デバッグには"g")、model file name、および、output file name (ここで、共有メモリドライバーの名前により、レイベルがプリフィックスされている。モデルをレイベルしたので、コンパイルされたモデルで使われているメモリを解放できますが、それは、fdeleteをコールすることで行ないます。このサブルーチンは、mmjobsに加え、mmsystemというモジュールをロードし、そこで得られます。

2.5 ランタイム・パラメータ

モデルをランしているときに、（すなわち、モデル自体を修正して、リコンパイルすることなく）Moselモデルのデータを修正する便利な方法は、*runtime parameters*を使う方法です。このようなパラメータは、モデルの最初に部分のparametersブロックでdeclareすることです。parameters

ブロックでは、すべてのパラメータにたいしてディフォルト値が与えられており、ディフォルト値以外の値が指定されていない場合は、モデルの実行時には、これらのディフォルト値が使われます。

下のモデルrtparams.mosについて考えてみてください。このモデルは、4種類のパラメータ（integer、real、string、Boolean）を受け取り、それらの値をプリントします。

```
model "Runtime parameters"
 parameters
  PARAM1 = 0
  PARAM2 = 0.5
  PARAM3 = ''
  PARAM4 = false
 end-parameters

 writeln(PARAM1, " ", PARAM2, " ", PARAM3, " ", PARAM4)

 end-model
```

Master model “ runrtparam.mos ” が、この（sub）modelを起動させるには、以下のモデルで行えます。一全てのランタイム　パラメータで新しい値が与えられます。

```
model "Run model rtparams"
 uses "mmjobs"

 declarations
  modPar: Model
 end-declarations
                                                ! Compile the model file
 if compile("rtparams.mos")<>0 then exit(1); end-if
 load(modPar, "rtparams.bim")                   ! Load the bim file
                                                ! Start model execution
 run(modPar, "PARAM1=" + 2 + ",PARAM2=" + 3.4 +
             ",PARAM3='a string'" + ",PARAM4=" + true)
 wait                                           ! Wait for model termination
 dropnextevent                                  ! Ignore termination event message

end-model
```

2.6 複数のサブモデルをランする

master model から、どのようにしてパラメータ化されたモデルをランするかを理解できれば、master model から、異なるサブモデルのインスタンスを実行するのは簡単です。次の二つの節で、サブモデルをsequential に、そして、parallel に実行することについて説明します。 説明を判りやすくするため、ここでの例では、サブモデルは、すべて、単一モデルのパラメータ化されたバージョンを使います。もちろん、単一のmaster model から、異なるサブモデルをコンパイルし、ロードし、ランすることもできます。

2.6.1 Sequential サブモデル

サブモデルのいくつかのインスタンスをシークエンシャルにランするには、下記の例で見るように(file runrtparamseq.mos)、単一モデルのインスタンスに使ったmaster modelに、小さな修正を加えるだけです。ものごとを単純にするため、各々の実行ごとに、単一のパラメータだけをリ

セットします。

```
model "Run model rtparams in sequence"
 uses "mmjobs"

 declarations
  A = 1..10
  modPar: Model
 end-declarations
                                                ! Compile the model file
 if compile("rtparams.mos")<>0 then exit(1); end-if
 load(modPar, "rtparams.bim")                   ! Load the bim file

 forall(i in A) do
  run(modPar, "PARAM1=" + i)                    ! Start model execution
  wait                                          ! Wait for model termination
  dropnextevent                                 ! Ignore termination event message
 end-do
end-model
```

サブモデルがコンパイルされ、一度、ロードされ、サブモデル・インスタンスの実行が始まると、現在、実行されているインスタンスが終了するまで、次のインスタンスの開始を待ちます。

2.6.2 Parallel サブモデル

サブモデルのインスタンスをパラレルに実行するには、もう少し、修正を行なうことが必要です。この場合も、サブモデルのコンパイルは、一回、行なえばよいのですが、しかし、インスタンスをパラレルでランする数だけ、ロードしなければなりません。今回は、サブモデルをすべて開始し、それらの終了を待つことになりますから、`wait` ステートメントを別個のループに移します。

```
model "Run model rtparams in parallel"
  uses "mmjobs"

  declarations
   A = 1..10
   modPar: array(A) of Model
  end-declarations
                                                ! Compile the model file
  if compile("rtparams.mos")<>0 then exit(1); end-if

  forall(i in A) do
   load(modPar(i), "rtparams.bim")              ! Load the bim file
   run(modPar(i), "PARAM1=" + i)                ! Start model execution
  end-do

  forall(i in A) do
   wait                                         ! Wait for model termination
   dropnextevent                                ! Ignore termination event message
  end-do

end-model
```

スクリーンにサブモデルのアウトプットが現れる順番は、決まっていません。なぜなら、モデル

がパラレルにランされているからです。しかし、サブモデルの実行は非常に早いので、よくわからないでしょう。サブモデルrtparams.mos のwritelnステートメントの直前に、wait(1)を加えてみてください。また、モデルの最初に、uses "mmjobs"ステートメントを加えることも必要です。これがすんだら、master modelのいくつかのランのアウトプットを比較してみてください。おそらく、ランごとに、アウトプットのシークエンスが異なっていることが判るでしょう。

2.7 異なるモデル間でのデータのやり取り

ランタイム・パラメータは、個々のデータの値をサブモデルに渡す方式の一つです。しかし、この方式は、サブモデルに、例えば、データ・テーブルやデータ・セットを渡す方式ではありません。また、これは、サブモデルから情報をリトリーブしたり、実行の最中に、モデル間でデータをやり取りする方式としては使えません。これらのタスクは、すべて、モジュール*mmjobs*で定義される二つのIO driver、すなわち、shmem driver、および、mempipe driverで対応します。前に述べたように、shmem driver は、one-to-many communication (one model writing, many reading) のためのものであり、mempipe driver は、many-to-one communicationのためのものです。one model writing、one model readingを行なう場合、どちらのドライバーも使えますが、shmemのほうが、概念的に使いやすいでしょう。

2.7.1 共有メモリドライバーを使う

shmem共有メモリドライバーを使って、データ・ブロックをメモリに書いたり、メモリから読んだりします。このドライバーの使い方は、物理的なファイルを使うときの方法とよく似ています。2.4節で、既に、その使い方の例を見ましたが、そこでは、filename はlabel で置き換えられ、"mmjobs.shmem:aLabel"のようなドライバーの名前がprefixされています。モデルに*mmjobs*モジュールがロードされていたり、メモリに、master modelのような、別のモデルがあるような場合は、短縮形"shmem:aLabel"を使うこともできます。

異なるモデル間でのデータをやりとりは、initializationsブロックを通して行なわれます。通常、shmem driver は、データをバイナリー形式でセーブするためにraw driver と結合されます。ここで、2.1節で見た最初のテスト・サブモデルの修正版であるtestsubshm.mosについて考えてみましょう。このモデルは、メモリからindex range を読み、resulting array をメモリに書き出します。

```
model "Test submodel"
  declarations
    A: range
    B: array(A) of real
  end-declarations

  initializations from "raw:"
    A as "shmem:A"
  end-initializations

  forall(i in A) B(i):= i^2
```

```
  initializations to "raw:"
    B as "shmem:B"
  end-initializations

end-model
```

このサブモデルをランするmasterモデルrunsubshm.mosは、下記のようなものになるでしょう。

```
model "Run model testsubshm"
  uses "mmjobs"

  declarations
    modSub: Model
    A = 30..40
    B: array(A) of real
  end-declarations
                                                ! Compile the model file
  if compile("testsubshm.mos")<>0 then exit(1); end-if
  load(modSub, "testsubshm.bim")                ! Load the bim file

  initializations to "raw:"
    A as "shmem:A"
  end-initializations

  run(modSub)                                   ! Start model execution
  wait                                          ! Wait for model termination
  dropnextevent                                 ! Ignore termination event message

  initializations from "raw:"
    B as "shmem:B"
  end-initializations

  writeln(B)

end-model
```

サブモデルのランが開始される前に、メモリにindex range が書かれ、ランが終わるとresult array をリトリーブして、master modelからプリントします。
メモリ･ブロックが使用されなくなったら、とくに、データ・ブロックが大きい場合、fdelete (subroutine provided by module *mmsystem*)をコールして、これらのブロックを解放することをお勧めします。なぜなら、これらのブロックは、それらのブロックを生成したモデルが終了しても、そして、明確にモデルがアンロードされたとしても、*mmjobs*モジュールが、Moselセッションの終了により、アンロードされるまで、残っているからです。このような理由から、master modelの終わりに、下記のステートメントを加えます。

```
fdelete("shmem:A")
fdelete("shmem:B")
```

2.7.2 メモリ･パイプドライバーを使う

メモリ･パイプIO ドライバーmempipe は､共有メモリドライバーについて見てきたこととは逆に

動きます。パイプは、そこに書くことができるようにするためには、まず、openされなければなりません。これを行なうには、initializations to の前に、initializations fromをコールします。サブモデル(file testsubpip.mos)は、次のようになるでしょう。

```
model "Test submodel"
  declarations
    A: range
    B: array(A) of real
  end-declarations

  initializations from "mempipe:indata"
    A
  end-initializations

  forall(i in A) B(i):= i^2

  initializations to "mempipe:resdata"
    B
  end-initializations

end-model
```

これは、確かに、前のサブモデルと大きく変わるものではありません。しかし、master model(file runsubpip.mos)の変更は、もっと大きいものです。すなわち、ここでは、インプット・データは、サブモデルが開始されてから、master modelによって書かれます。master modelは、次いで、サブモデルがその実行を終了する前に、結果のデータを読むための新しいパイプをopenします。

```
model "Run model testsubpip"
  uses "mmjobs"

  declarations
    modSub: Model
    A = 30..40
    B: array(A) of real
  end-declarations

                                          ! Compile the model file
  if compile("testsubpip.mos")<>0 then exit(1); end-if
  load(modSub, "testsubpip.bim")          ! Load the bim file

  run(modSub)                             ! Start model execution

  initializations to "mempipe:indata"
    A
  end-initializations

  initializations from "mempipe:resdata"
    B
  end-initializations

  wait                                    ! Wait for model termination
  dropnextevent                           ! Ignore termination event message

  writeln(B)
  ]
end-model
```

ファイルが、読むためにパイプをオープンすると、このパイプを通して、要求したデータを受け取るまで、ブロックされたままの状態のままです。したがって、このケースの場合、プログラム・コントロールのフローは、下記のようになります。

1. master modelは、サブモデルをスタートする。
2. サブモデルは、インプット・データ・パイプをオープンして、master modelが、それに、書くのを待つ。
3. インプット・データが渡されるとサブモデルは実行を続け、master modelは、アウトプット・データ・パイプをオープンして、アウトプットが渡されるのを待つ。
4. アウトプット・データ・パイプがオープンされると、サブモデルは、アウトプット・データ・パイプに書き、終了する。
5. master modelは、結果をプリントする。

3 カラムの生成：異なるモデルを、順番に解く

この章で使うcutting stock example は、‘Mosel User Guide’記載のものです。カラム生成のアルゴリズムと、それをMoselでどのように行なったらよいかについては、読者の皆さんは、このマニュアルを参照してください。

カラム生成のアルゴリズムは、たくさんの変数を持つ線形計画モデルでよく使われます。たくさんの変数があると、問題マトリックスのすべてのカラムをはっきりと示すのが困難だからです。最初、カラムの数を制限したマトリックスから始め、問題のソリューションを得てから、カラム生成アルゴリズムを使って、手持ちのソリューションを改善するカラムを追加して行きます。

cutting stock問題のカラム生成アルゴリズムでは、新しいカラム(= cutting pattern)を決定するのに、現在のソリューションのdual valueに基づき、knapsack問題を解く必要があります。User Guideのインプリメンテーションと下記の相違は、このknapsack（サブ）問題を扱う方法が異なる、ということにあります。User Guideのインプリメンテーションでは、Moselの*constraint hiding functionality*が使い、制約式の部分集合を滑らかにします（blend out）が、下記では、モデルの中で、サブ問題が、そのままで実行されます。この両方が実行することは、まったく同一のアルゴリズムで、また、パーフォーマンスも同程度です。しかし、インスタンスの数が多いと、two-model 方式の方が、少し、効率性がよいようです。なぜなら、各モデルは、(un)hidden constraintsを選択することなく、正確に、解くべき問題を定義するからです。

ここの例では、問題に加える変化は、すべての最適化のランごとに、問題を完全に、再度、ロードすることが必要になるような変化です。このマルチモデル方式の利点は、main (cutting stock) problemへ加えられる変化(bound updates)が小さく、すべての最適化のランごとに、この問題を、再度、ロードすることが必要でない場合に、明白に現れてきます。

3.1 例題: **cutting stock**

ある製紙工場では、固定幅*MAXWIDTH* のロールを生産し、その後、顧客オーダーに基づき、小さなロールにカットしています。ロールは、*NWIDTHS* だけの種類の異なるサイズにカットできます。各サイズ i ($DEMAND_i$)ごとのオーダーは、需要として与えられます。この製紙工場の目的関数は、需要を、できるだけ少ない数の紙ロールで満たし、ロスを最小化することです。

ロールの数を最小にするという目的関数は、現在の手持ちの需要を満たすのに最も良いカット・パターンの部分集合を選択する、として表現できます。手動で、どのようにしたら、すべての可能なカット・パターンを計算したらよいかは定かではないので、パターンの基本セット ($PATTERNS_1, \ldots, PATTERNS_{NWIDTH}$) から始めます。これは、大きい元のロールから、同じ幅の、各小さいロールを、できるだけ多くカットするパターンとなっています。ここで変数use_jを定義して、カット・パターンj ($j \in$ WIDTHS = {1, … ,*NWIDTH*})が使われる回数を示すとすると、目的関数は、これらの変数の合計となり、制約式は、各サイズへの需要が満たされることとして定式化されます。

$$\text{minimize} \sum_{j \in WIDTHS} use_j$$

$$\text{Subject to} \sum_{j \in WIDTHS} PATTERNS_{ij} \cdot use_j \geq DEMAND_i$$

$$\forall j \in WIDTHS : use_j \leq [DEMAND_j / PATTERN_{ij}], use_j \in IN$$

この製紙工場は、カット・パターンの基本セットで、需要を満たせますが、やり方によっては、大きなロールを必要以上に使うこと、および、オーダー数量以上の小さなロールを生産してしまうことで大きなムダを生み出してしまうこともありえます。したがって、カラム生成ヒューリスティックを使い、ムダを小さくするカット・パターンを見つけることが必要です。
その行ない方は、MIP サーチを行なう前に、トップ・ノードで、ヒューリスティックにカラム生成ループを行なうことです。カラム生成ループを繰り返すごとに、下記のステップを行ないます。

1. LP問題を解き、ベイシスをセーブする
2. ソリューションの値を得る
3. 手許にある現在のソリューションに基づき、より有利なカット・パターンを計算する
4. 新しいカラム(= cutting pattern)を生成する：目的関数、および、対応する需要制約式に項を追加する
5. 修正した問題とセーブしておいたベイシスをロードする

このループのステップ3 では、下記のインテジャーknapsack問題を解くことが必要です。

$$\text{maximize z} = \sum_{j \in WIDTHS} C_i \cdot x_j$$

$$\sum_{j \in WIDTHS} A_j \cdot x_j \leq B$$

$$\forall_j \in WIDTHS : x_j \text{ integer}$$

この二番目の最適化問題は、主問題であるcutting stock 問題とは変数を共有していないので、主問題とは独立しています。

3.2 インプリメンテーション

インプリメンテーションは、二つの部分かなっています。すなわち、「cutting stock problem の定義、および、カラム生成アルゴリズムを持つmaster model (file paperp.mos)」、そして、masterからランされる「knapsack model (file knapsack.mos)」です。

3.2.1 Master model

cutting stockモデルの主要部分は、下記のようなものです。

```
model "Papermill (multi-prob)"
  uses "mmxprs", "mmjobs"

  forward procedure column_gen
  forward function knapsack(C:array(range) of real,
                            A:array(range) of real,
                            B:real,
                            xbest:array(range) of integer): real

  declarations
   NWIDTHS = 5                                  ! Number of different widths
   WIDTHS = 1..NWIDTHS                          ! Range of widths
   RP: range                                    ! Range of cutting patterns
   MAXWIDTH = 94                                ! Maximum roll width
   EPS = 1e-6                                   ! Zero tolerance

   WIDTH: array(WIDTHS) of real                 ! Possible widths
   DEMAND: array(WIDTHS) of integer             ! Demand per width
   PATTERNS: array(WIDTHS, WIDTHS) of integer ! (Basic) cutting patterns

   use: array(RP) of mpvar                      ! Rolls per pattern
   soluse: array(RP) of real                    ! Solution values for variables ‘use’
   Dem: array(WIDTHS) of linctr                 ! Demand constraints
   MinRolls: linctr ! Objective function

   Knapsack: Model                              ! Reference to the knapsack model
  end-declarations

  WIDTH:: [ 17, 21, 22.5, 24, 29.5]
  DEMAND:: [150, 96, 48, 108, 227]

                                                ! Make basic patterns
  forall(j in WIDTHS) PATTERNS(j,j) := floor(MAXWIDTH/WIDTH(j))

  forall(j in WIDTHS) do
   create(use(j))                               ! Create NWIDTHS variables ‘use’
   use(j) is_integer                            ! Variables are integer and bounded
   use(j) <= integer(ceil(DEMAND(j)/PATTERNS(j,j)))
  end-do

  MinRolls:= sum(j in WIDTHS) use(j)            ! Objective: minimize no. of rolls
                                                ! Satisfy all demands
  forall(i in WIDTHS)
   Dem(i):= sum(j in WIDTHS) PATTERNS(i,j) * use(j) >= DEMAND(i)

  res:= compile("knapsack.mos")                 ! Compile the knapsack model
  load(Knapsack, "knapsack.bim")                ! Load the knapsack model
  column_gen                                    ! Column generation at top node

  minimize(MinRolls)                            ! Compute the best integer solution
                                                ! for the current problem (including
                                                ! the new columns)
  writeln("Best integer solution: ", getobjval, " rolls")
  write(" Rolls per pattern: ")
  forall(i in RP) write(getsol(use(i)),", ")
  writeln
```

```
end-model
```

カラム生成ループごとに、新しいデータでランだけすればよいように、カラム生成ヒューリスティックを開始する前に、knapsack モデルがコンパイルされ、ロードされます。（手順column_genは、User Guide で示されている例と同じですので、ここでは、省略します。）knapsack モデルは、function knapsackからランされますが、function knapsackは、そのパラメータとして、knapsack 問題とそのソリューションの値のデータを使います。function knapsackは、すべてのデータを共有メモリにセーブし、次いで、knapsackモデルのランを行なって、共有メモリからソリューションをリトリーブします。返す値は、knapsack 問題の目的関数の値(zbest)です。

```
function knapsack(C:array(range) of real,
                  A:array(range) of real,
                  B:real,
                  xbest:array(range) of integer):real

  initializations to "raw:noindex"
   A as "shmem:A" B as "shmem:B" C as "shmem:C"
  end-initializations

  run(Knapsack, "NWIDTHS="+NWIDTHS)      ! Start solving knapsack subproblem
  wait                                   ! Wait until subproblem finishes
  dropnextevent                          ! Ignore termination message

  initializations from "raw:"
   xbest as "shmem:xbest" returned as "shmem:zbest"
  end-initializations

end-function
```

二つのモデルを、順繰りに行なうには、（master modelを継続するには、knapsack問題の結果をリトリーブする必要があるので）、runステートメントのすぐ後ろに、手順waitへのコールを追加しなければなりません。そうしておかないと、master modelの実行は、child modelとコンカレントリーに継続してしまいます。child modelは、終了すると、‘termination’ event (an event of class EVENT_END)を送ります。しかし、アルゴリズムは、このevent を必要としないので、dropnexteventをコールして、モデルのevent queue から、このeventを取り除きます。

3.2.2 Knapsack model

knapsackモデルの実行は、簡単でステートメント。問題データを、すべて、共有メモリから取り、問題を解いてから、ソリューションを共有メモリにセーブします。

```
model "Knapsack"
  uses "mmxprs"

  parameters
   NWIDTHS=5                             ! Number of different widths
  end-parameters

  declarations
```

```
  WIDTHS = 1..NWIDTHS                    ! Range of widths
  A,C: array(WIDTHS) of real             ! Constraint + obj. coefficients
  B: real                                ! RHS value of knapsack constraint
  KnapCtr, KnapObj: linctr               ! Knapsack constraint+objective
  x: array(WIDTHS) of mpvar              ! Knapsack variables
  xbest: array(WIDTHS) of integer        ! Solution values
end-declarations

initializations from "raw:noindex"
  A as "mmjobs.shmem:A" B as "mmjobs.shmem:B" C as "mmjobs.shmem:C"
end-initializations

                                         ! Define the knapsack problem
  KnapCtr:= sum(j in WIDTHS) A(j)*x(j) <= B
  KnapObj:= sum(j in WIDTHS) C(j)*x(j)

forall(j in WIDTHS) x(j) is_integer

                                         ! Solve the problem and retrieve the solution
  maximize(KnapObj)
  z:=getobjval
  forall(j in WIDTHS) xbest(j):=round(getsol(x(j)))

  initializations to "raw:"
    xbest as "mmjobs.shmem:xbest" z as "mmjobs.shmem:zbest"
end-initializations

end-model
```

このモデルでは、共有メモリドライバー shmem を、モジュールの名前*mmjobs*でprefixしてあります。このようにするのは、knapsack問題を切り離してランしたい、すなわち、メモリに*mmjobs*モジュールをロードするcutting stock master modelなしでランしたい場合のみです。
ドライバーにモジュールの名前を明示的に加えなければならないケースは、それぞれに異なるモジュールで定義された、いくつかのshmemドライバーを区別しなければならないときです。

3.3 結果

上のモデルのデータを使い、カラム生成アルゴリズムは、cutting stock問題のLP緩和の値をもともとの177.67から160.95に降ろし、6つの新しいパターンを生成します。MIPは、下記のパターンを使って、161ロールというソリューションを見つけます。

Widths						
Pattern	**17**	**21**	**22.5**	**24**	**29.5**	**Usage**
3	**0**	**0**	**4**	**0**	**0**	**1**
5	**0**	**0**	**0**	**0**	**3**	**15**
6	**0**	**1**	**0**	**3**	**0**	**32**
8	**2**	**0**	**0**	**0**	**2**	**75**
10	**0**	**2**	**1**	**0**	**1**	**32**
11	**0**	**0**	**2**	**2**	**0**	**6**

4 いくつかのモデル・インスタンスをパラレルに解く

この章では、どのようにして、いくつかのモデルをパラレルに実行し、これらのモデル間で、ソリューション情報をやり取りするかについて説明します。この仕組みは、Moselをマルチプロセッサー・マシンで使う、つまり、利用可能なプロセッサーの数に対応する数のモデルを解こうとするような場合、特に、興味深いものです。

ここでのアイディアは、（ソリューション・アルゴリズムのパラメータ設定のみが異なる）同一のMIPモデルのいくつかのインスタンスをコンカレントリーにランし、最初のモデルが終了したときに、すべてのランを停止するというアイディアです。異なるソリューション・アルゴリズムが、あるアルゴリズムは（良い）ソリューションを素早く見つける、別のソリューション・アルゴリズムは、最も良いソリューションが見つかったとき、オプティマリティの判定に優れているというような意味で、相互補完的であるとするならば、MIPサーチの間に、ソリューション上を交換することによるシナジー効果を期待できると考えてもよいでしょう。

この仕組みを実行するには、まず、master model とparameterizable child modelを定義します。master modelは、モデルのランを開始し、ソリューションの更新を調整、統合し、また、child modelは、ロードされ、好きなだけの変形（version）でランされます。child modelは、すべて、同一のソルバー(Xpress-Optimizer)を使いますが、別のソルバーも、そのソルバーがサーチとインタラクトするのに必要な機能を定義できれば、使うことができます。

4.1 例題：economic lot sizing

Economic lot sizing (ELS)では、所与の計画期間を対象に生産計画を作成します。この例では、計画期間は$TIMES = 1, \cdots, T$です。期間tの需要は$DEMAND_{pt}$、製品の種類はp $(p \in PRODUCTS)$で、需要$DEMAND_{pt}$は、この期間の生産と、前の期間からの繰越在庫で満たされなければなりません。

段取りコスト$SETUPCOST_t$はその期間の生産量と関連し、各期間の生産キャパシティCAP_tには制限があります。在庫保有のコストはありません。

期間tでの、製品pの生産量を示すデシジョン変数$produce_{pt}$、および、期間tで製品pの段取りが発生するかどうかを示すバイナリー変数$setup_{pt}$を導入します。このバイナリー変数は、段取りがあるときは($setup_{pt} = 1$) 、ないときは($setup_{pt} = 0$)となります。

この問題の数学的なモデルは下記のようになります。

$$\text{minimize} \quad \sum_{t \in TIMES} (SETUPCOST_t \cdot \sum_{p \in PRODUCTS} setup_{pt} + \sum_{p \in PRODUCTS} PRODCOST_{pt} \cdot produce_{pt})$$

$$\forall p \in PRODUCTS, t \in TIMES : \sum_{s=1}^{t} produce_{ps} \geq \sum_{s=1}^{t} DEMAND_{ps}$$

$$\forall p \in PRODUCTS, t \in TIMES : produce_{pt} \leq D_{ptT} \cdot setup_{pt}$$

$$\forall t \in TIMES : \sum_{p \in PRODUCTS} produce_{pt} \leq CAP_t$$

$$\forall p \in PRODUCTS, t \in TIMES : setup_{pt} \in \{0,1\}, produce_{pt} \geq 0$$

目的関数は、総コストを最小にすることです。2行目の制約式で、期間0かたら期間 *t* までの製品 *p* の生産が、この期間の製品 *p* の需要を満たさなければならないことを定式化します。それに続く制約式は、「期間 *t* に生産が行なわれれば、期間 *t* には段取りもある」こと、ここで、D_{ptl} は、期間 *t* かたら期間 *l* までの製品 *p* の需要を意味します。また、期間 *t* の生産キャパシティには制限があること、そして、変数 $setup_{pt}$ はバイナリー変数であることを意味しています。

4.1.1 Cutting plane algorithm

ELS でよく知られた不等式は、いわゆる (*l*, *S*)-inequalities [PW94]です。D_{ptl} が期間 *t* から期間 *l* までの製品 *p* の需要を意味するき、各期間 *l*、および、期間1かたら期間 *l* の各部分集合期間 *S* に対して、(*l*, *S*)-inequalityは、下記のようになります。

$$\sum_{\substack{t=1 \\ t \in s}}^{l} produce_{pt} + \sum_{\substack{t=1 \\ t \notin s}}^{l} D_{ptl} \cdot setup_{pt} \geq D_{p1l}$$

この不等式は、*S* に含まれる期間の実際の生産 $produce_{pt}$ ＋残りの期間（*S* に含まれていない期間）の潜在的な最大生産 $D_{ptl} \cdot setup_{pt}$ は、少なくとも、期間1かたら期間 *l* までの総需要に等しくなければならないということを言っています。
これらの (*l*, *S*)-inequalities に基づき、下記のcutting plane algorithm を作成することができます。

1. LP問題を解く
2. 下記の不等式を調べて、(*l*, *S*)-inequalitiesの違反を識別する

$$\sum_{t=1}^{l} \min(produce_{pt}, D_{pti} \cdot setup_{pt}) \geq D_{p1l}$$

3. 違反している（満たされていない）不等式を、新たなcutとして問題に加える
4. LP問題を、再度、解く

このアルゴリズムをどのように作るかについては、無数のオプションがあります。例えば、

- Cutをルート・ノードの中のみに生成するか、もしくは、サーチの中でも生成するか (Cut-and-Branch vs Branch-and-Cut)
- ノードで生成するcut generation passの数（例えば、passを一つ、もしくは、上の2. から4.のステップをループさせ、cutが生成されないようになるまで繰り返す）

- cut生成を、サーチ･ツリーの中で、どれくらいまで深く行なうか（ある程度の深さまでか、それとも、すべてのノード？）
- *(l, S)*-cutsだけを使うか、それとも、他のもの（例えば､ソルバーにより生成されるdefault cuts）と組み合わせるか

下記の*(l, S)*-cut生成アルゴリズムの実行では、トップ・ノードのみでcut生成を(TOPONLY = true)、一つ、もしくは、いくつかのrounds of cuts (SEVERALROUNDS = true)を行なうように設定しましょう。

4.2 インプリメンテーション

*mmjobs*では、eventは、あるchildから別のchildに、直接、送られることはなく、必ず、「parentとchildの対」の間でやりとりします。したがって、「solution found」というメッセージは、parent modelに送られる必要があり、次いで、parent modelは、このメッセージをその他のすべてのchild modelに送ります。
ここで強調しておきたいもう一つの点は、ELS model fileのコンパイルは1回のみ行われるということ、しかし、メモリにロードされるインスタンスの数は、ランしたいchild modelの数と対応していなければならないということです。

4.2.1 Master model

master modelは、child modelをコンパイルし、ロードし、ランをします。そして、ソリューションの更新を調整します。ソリューションの更新には、注意が必要です。なぜなら、新しいソリューションは、必ずしも、child modelにより、これまでに見つけられ、報告されているソリューションよりも良いとは限らないからです。
例えば､2つのモデルが、ほとんど同時に、ソリューションを見つけたとすると、master modelに早く着いたソリューションがよりよいソリューションかも知れません。そのような場合、早く着いたソリューションが、続いてきたソリューションにより、書き換えられてはいけません。
ランの終わりにソリューション・ディスプレーへの表示を良いものにするために、master modelは、problem data from fileの一部を読み込みます。

```
model "Els master"
  uses "mmjobs"

  parameters
    DATAFILE = "els5.dat"
    T = 45
    P = 4
  end-parameters

  declarations
    RM = 0..5                          ! Range of models
    TIMES = 1..T                       ! Time periods
    PRODUCTS = 1..P                    ! Set of products
```

```
    solprod: array(PRODUCTS,TIMES) of real     ! Sol. values for var.s produce
    solsetup: array(TIMES) of real             ! Sol. values for var.s setup
    DEMAND: array(PRODUCTS,TIMES) of integer ! Demand per period

    modELS: array(RM) of Model                 ! Models
    modid: array(set of integer) of integer    ! Model indices
    NEWSOL = 2                                 ! Identifier for "sol. found" event
    Msg: Event                                 ! Messages sent by models
  end-declarations

! Compile, load, and run models M1 and M2
  M1:= 1; M2:=2
  res:= compile("elsp.mos")
  load(modELS(M1), "elsp.bim")
  load(modELS(M2), "elsp.bim")
  forall(m in RM) modid(getid(modELS(m))):= m
  run(modELS(M1), "ALG="+M1+",DATAFILE="+DATAFILE+",T="+T+",P="+P)
  run(modELS(M2), "ALG="+M2+",DATAFILE="+DATAFILE+",T="+T+",P="+P)

  objval:= MAX_REAL
  algsol:= -1; algopt:= -1

  repeat
    wait                                       ! Wait for the next event
    Msg:= getnextevent                         ! Get the event
    if getclass(Msg)=NEWSOL then               ! Get the event class
      if getvalue(Msg) < objval then           ! Value of the event (= obj. value)
        algsol:= modid(getfromid(Msg))         ! ID of model sending the event
        objval:= getvalue(Msg)
        writeln("Improved solution ", objval, " found by model ", algsol)
        forall(m in RM | m <> algsol) send(modELS(m), NEWSOL, objval)
  else
        writeln("Solution ", getvalue(Msg), " found by model ",
                modid(getfromid(Msg)))
        end-if
    end-if
  until getclass(Msg)=EVENT_END ! A model has finished

  algopt:= modid(getfromid(Msg))               ! Retrieve ID of terminated model
  forall(m in RM) stop(modELS(m))              ! Stop all running models

! Retrieve the best solution from shared memory
    initializations from "raw:noindex"
      solprod as "shmem:solprod"+algsol
      solsetup as "shmem:solsetup"+algsol
    end-initializations

    initializations from DATAFILE
      DEMAND
    end-initializations

  ! Solution printing
  writeln("Best solution found by model ", algsol)
  writeln("Optimality proven by model ", algopt)
  writeln("Objective value: ", objval)
  write("Period setup ")
  forall(p in PRODUCTS) write(strfmt(p,-7))
  forall(t in TIMES) do
    write("¥n ", strfmt(t,2), strfmt(solsetup(t),8), " ")
    forall(p in PRODUCTS) write(strfmt(solprod(p,t),3), " (",DEMAND(p,t),")")
  end-do
  writeln
```

```
end-model
```

ここでのインプリメンテーションでは、array modidを定義します。これは、モデルで使われているmodel index と、Moselの中のモデル内部IDの間の通信のやり取りを確立するためのものです。child modelがmaster modelにeventを送ると、function getfromidにより、かならず、そのIDがリトリーブされ、対応するmodel indexがストアされ、これを使って、後で、ソリューションのプリントを行ないます。

4.2.2 ELS model

ELSのchild modelは、モデルが分離して実行できるように書かれます。特に、モデルは、それぞれ、ファイルからのデータの完全なinitializationを行ないますが、これは、master modelの大きな効率性を保持するためで、データは共有メモリ経由で、child modelに渡されます。（しかし、ここでの例では、データ・ハンドリングの時間は、ソリューション・アルゴリズムのランに必要な時間と較べ、ごく僅かです。）
ELS modelの主要部分には、モデル自体の定義とソリューション・アルゴリズムの選択が入っています。

```
model Els
  uses "mmxprs","mmjobs"

  parameters
    ALG = 0                                   ! Default algorithm: no user cuts
    DATAFILE = "els4.dat"
    T = 60
    P = 4
  end-parameters

  forward procedure tree_cut_gen
  forward public function cb_node: boolean
  forward public function cb_updatebnd(node:integer): integer
  forward public procedure cb_intsol

  declarations
    NEWSOL = 2                                ! "New solution" event class
    EPS = 1e-6                                ! Zero tolerance
    TIMES = 1..T                              ! Time periods
    PRODUCTS = 1..P                           ! Set of products

    DEMAND: array(PRODUCTS,TIMES) of integer ! Demand per period
    SETUPCOST: array(TIMES) of integer        ! Setup cost per period
    PRODCOST: array(PRODUCTS,TIMES) of real ! Production cost per period
    CAP: array(TIMES) of integer              ! Production capacity per period
    D: array(PRODUCTS,TIMES,TIMES) of integer ! Total demand in periods t1 - t2

    produce: array(PRODUCTS,TIMES) of mpvar  ! Production in period t
    setup: array(TIMES) of mpvar              ! Setup in period t

    solprod: array(PRODUCTS,TIMES) of real    ! Sol. values for var.s produce
    solsetup: array(TIMES) of real            ! Sol. values for var.s setup

    Msg: Event                                ! An event
```

```
      end-declarations

      initializations from DATAFILE
        DEMAND SETUPCOST PRODCOST CAP
      end-initializations

      forall(p in PRODUCTS,s,t in TIMES) D(p,s,t):= sum(k in s..t) DEMAND(p,k)

! Objective: minimize total cost
      MinCost:= sum(t in TIMES) (SETUPCOST(t) * setup(t) +
                                 sum(p in PRODUCTS) PRODCOST(p,t) * produce(p,t) )

! Production in period t must not exceed the total demand for the
! remaining periods; if there is production during t then there
! is a setup in t
      forall(t in TIMES)
        sum(p in PRODUCTS) produce(p,t) <= sum(p in PRODUCTS) D(p,t,T) * setup(t)

! Production in periods 0 to t must satisfy the total demand
! during this period of time
      forall(p in PRODUCTS,t in TIMES)
      sum(s in 1..t) produce(p,s) >= sum (s in 1..t) DEMAND(p,s)

! Capacity limits
      forall(t in TIMES) sum(p in PRODUCTS) produce(p,t) <= CAP(t)

      forall(t in TIMES) setup(t) is_binary          ! Variables setup are 0/1

      setparam("zerotol", EPS/100)                   ! Set Mosel comparison tolerance
      SEVERALROUNDS:=false; TOPONLY:=false

      case ALG of
      1: do
        setparam("XPRS_CUTSTRATEGY", 0)    ! No cuts
        setparam("XPRS_HEURSTRATEGY", 0)   ! No heuristics
      end-do
      2: do
        setparam("XPRS_CUTSTRATEGY", 0)    ! No cuts
        setparam("XPRS_HEURSTRATEGY", 0)   ! No heuristics
        setparam("XPRS_PRESOLVE", 0)       ! No presolve
      end-do
      3: tree_cut_gen                                ! User branch&cut (single round)
      4: do                                          ! User branch&cut (several rounds)
        tree_cut_gen
        SEVERALROUNDS:=true
      end-do
      5: do                                          ! User cut&branch (several rounds)
        tree_cut_gen
        SEVERALROUNDS:=true
        TOPONLY:=true
      end-do
    end-case

! Parallel setup
      setcallback(XPRS_CB_PRENODE, "cb_updatebnd") ! Node pre-treatment callback
      setcallback(XPRS_CB_INTSOL, "cb_intsol")  ! Integer solution callback
      setparam("XPRS_SOLUTIONFILE",0)           ! Do not save solutions to file

! Solve the problem
      minimize(MinCost)

end-model
```

手順tree_cut_genにより、ユーザのcut生成ルーチンがセットアップできます。これは、branch-and-bound searchのトップノードにおいてのみcutを生成するか(TOPONLY) 、ノードあたり、いくつかのcutを生成するか(SEVERALROUNDS)のどちらかです。ここでは、cut生成ルーチンcb_node自体の定義は省かれています。

```
procedure tree_cut_gen
   setparam("XPRS_HEURSTRATEGY", 0)        ! Switch heuristics off
   setparam("XPRS_CUTSTRATEGY", 0)         ! Switch automatic cuts off
   setparam("XPRS_PRESOLVE", 0)            ! Switch presolve off
   setparam("XPRS_EXTRAROWS", 5000)        ! Reserve extra rows in matrix

   setcallback(XPRS_CB_CUTMGR, "cb_node")  ! Define the cut manager callback
end-procedure
```

コンカレントにランされているchild model間の情報のやり取りは、2つの部分を持っています。すなわち、(a)見つけられたインテジャー・ソリューションは、セーブされ、master modelに伝えられなければならない。(b)master modelから送られるbound updates は、サーチに組み込まれなければならない、です。Xpress-Optimizer は、ユーザ構造の中にソリューションをセーブするための固有な*integer solution callback*を提供しています。An obvious place for bound updates in nodes is the *cut-manager callback* function. ノードでのbound updatesの場所は、*cut-manager callback* functionです。しかし、このfunctionが、もし、アルゴリズムの何らかの設定で、既に、使われてしまっているときは、すべてのノードで呼び出せる別のcallback function、すなわち、*node pre-treatment callback*を使います。

```
! Update cutoff value
   public function cb_updatebnd(node:integer): integer
      if not isqueueempty then
         repeat
            Msg:= getnextevent
         until isqueueempty
         newcutoff:= getvalue(Msg)
         setparam("XPRS_MIPABSCUTOFF", newcutoff)
         if (newcutoff < getparam("XPRS_LPOBJVAL")) then
            returned:= 1                          ! Node becomes infeasible
   end-if
  end-if
end-function

! Store and communicate new solution
  public procedure cb_intsol
   objval:= getparam("XPRS_LPOBJVAL")          ! Retrieve current objective value
   cutoff:= getparam("XPRS_MIPABSCUTOFF")
   if(cutoff > objval) then
   setparam("XPRS_MIPABSCUTOFF", objval)
   end-if

! Get the solution values
   forall(t in TIMES) do
    forall(p in PRODUCTS) solprod(p,t):=getsol(produce(p,t))
    solsetup(t):=getsol(setup(t))
   end-do
```

```
! Store the solution in shared memory
   initializations to "raw:noindex"
     solprod as "shmem:solprod"+ALG
     solsetup as "shmem:solsetup"+ALG
   end-initializations

! Send "solution found" signal
   send(NEWSOL, objval)
end-procedure
```

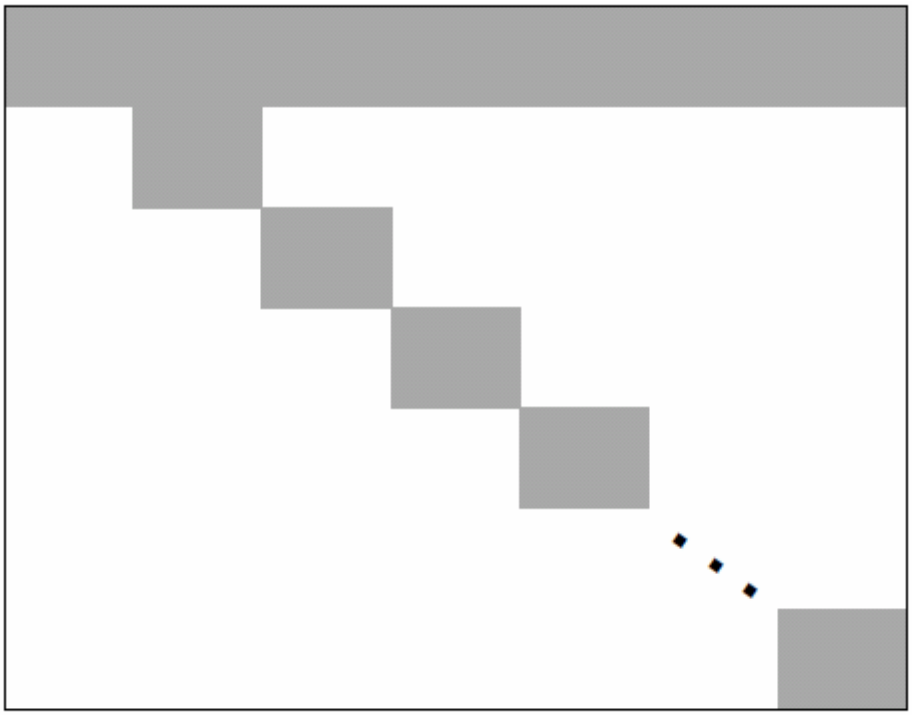

Figure 1:主要部分が、アンギュラー構造を持つ係数マトリックス」

bound update callback function は、event queueにeventが入っているかどうかをチェックします。そして、event が入っていると、event queueのeventを、すべて、取り出して、最後のeventの値を新しいcutoff valueとします。event queue を空にするループの背後にある論拠は、master modelは、最後にチェックした時点以後に、いくつかの改善されたソリューション値を送信してきているかもしれないが、最も良い値は最後に送信されてきたものである、すなわち、event queue の最後の値だ、ということです。*integer solution callback*は、共有メモリにソリューションの値を書き、identifier of the model (= value of ALG)を加えます。これにより、同時に、ソリューションの値を書き込んだかも知れない2つのchild modelが、同意ツリーのメモリ領域を使わないようにします。

4.3 Results

2つのモデルを使って行なったランは、下記のようなログを生成するでしょう。ここで、サーチ

を終わらせたモデルは、最適解を見つけたモデルとは異なることに注目してください。

```
Improved solution 1283 found by model 2
Improved solution 1250 found by model 2
Improved solution 1242 found by model 1
Improved solution 1236 found by model 2
Improved solution 1234 found by model 2
Best solution found by model 2
Optimality proven by model 1
Objective value: 1234
```

5 Dantzig-Wolfe decomposition: sequential と parallel 解法を組み合わせる

Dantzig-Wolfe decomposition (詳しくは [Teb01] をお読みください)は、一つの解法です。この解法は、比較的、少数の制約式を取り除いてやると、問題が、それぞれ独立した問題に分解できるような問題で有効な解法です。このことは、Figure 1のように、制約式マトリックスの行と列を並べ替えることができることを意味し、そこでは、ノンゼロ係数は、灰色の領域のみに位置します。このような*primal block angular structure* は、Xpress-IVEで問題マトリックスを視覚化すると、即座に、はっきり見えるようになります。しかし、多くの場合、期間、製品、工場の位置などのようなcommon index (sub)setでグループ化して、制約式の定義を組み立てなおすことが必要になるでしょう。

いくつかの、もしくは、すべてのsubproblemの変数を含む制約式（目的関数を含む）は、*global constraints* (also: common, linking, or master constraints)と呼ばれます。これらの制約式は、*master problem*を定式化するのに使われます。係数マトリックスの対角線上にある個々のブロックは、master problemの統制の下に、*pricing subproblems*として解かれます。そして、master problem を解くことにより、もともとの問題の解を得ます。master problem は、(pricing problemsの基本的な実行可能解とun-bounded directionsの集合で定義される)たくさんの変数を持つので、私たちは、これらの変数の小さな部分集合を対象に、*restricted master problem* を使います。restricted master problemの変数の「active set of variablesに入れる変数」は、pricing subproblemsを解くことで決められます。restricted master problemのdual valueに基づくpricing problemの目的関数により、各extreme pointでの目的関数の値は、そのextreme pointに対応するmaster problemの変数のreduced cost (or price)になります。

最大化問題では、修正pricing problemを解くことにより、maximum reduced costを持つ基本実行可能解が生成されます。あるextreme pointでの目的関数の値が正ならば、master problemに、対応するmaster problem変数が追加されます。そして、すべてのextreme pointでのminimum objective valueが負であるならば、master problemの変数で、その時点でのmaster problemのソリューションを改善できる変数は存在しません。

Dantzig-Wolfe decomposition の計算上の利点は、pricing problemの大量の計算量をこなすときに生れます。pricing problemは、もともとの問題よりも大幅に小さく、したがって、解くのが容易になります。この小論のテーマとの関連で興味深い一つの側面は、subproblemが相互に独立であることで、したがって、subproblemを同時並行的に解くことができる、ということです。このdecompositionによるアプローチの潜在的な欠点は、master problemのサイズがとても大きいことです。master problemは、もともとの問題よりも制約式の数は少ないのですが、変数の数が大幅に多くなります。通常、すべての変数を明示的に生成することは求められませんが、master problem の実行可能領域は、もともとの問題よりも複雑なので、ソリューション・パスが長くなります。さらに、master problemの変数をダイナミックに生成することから、数値計算上の問題（numerical problems）が発生することがあります。

decompositionアプローチのパーフォーマンスには、多数の要因が影響します。したがって、特

定のアプリケーションごとに、このソリューションが適しているかどうかを実験的に計算してみることが必要でしょう。このようなテストには、与えられた問題を異なった方法で分解してみることも含まれています。問題の分解の定義ための一般的なルールの一つは、global constraintsを少なくすることを狙うべきである、ということです。なぜなら、master problemを素早く解けることが重要だからです。さらに、pricing problemは、degeneracyによる計算上の問題を回避するため、pricing problem それ自体で、良く定式化された問題であるように構築されるべきです。

5.1 例題: multi-item, multi-period production planning

Coco社は、二つの種類の粉末のココアの素を生産できる工場を二つ持っています。最初の工場は、1ヶ月あたり、合計400トンのキャパシティを持ち、二番目の工場は、合計500トンのキャパシティを持っています。マーケッティング部門は、次の4ヶ月で売れる製品別の最大販売量の推定値、および、予想販売価格を生産部門に寄越します。生産にはいくつかの原材料が必要ですが、最終製品1トンあたりに必要な原材料所要量は判っています。期末に、最終製品、および、原材料を工場で貯蔵しておき、次の期に繰り越せますが、その保有には、1期」、1トンあたり、一定のコストが発生します。工場での原材料の貯蔵能力は、最大300トンです。利益を最大にするには、この2つの工場を、どのように操業したらよいでしょうか。

5.1.1 もともとのモデル

それぞれ、*PRODS*で最終製品の、*FACT*で工場の、*RAW*で原材料の、そして、$TIMES = \{1, \ldots, NT\}$で対象とする期間の集合を表すものとします。

また、$make_{pft}$というデシジョン変数を定義して、製品pの、工場fにての、期間tでの生産量を表すものとします。さらに、ある期から次の期への移行を定式化し、異なるタイプのコストを考慮するため、いくつかの変数を追加します。$sell_{pft}$ で、製品pの、工場fでの、期間tでの販売量を表し、buy_{rft}で、原材料rの、工場fで購入される、期間tでの購入量を表し、また、それぞれ、期間$t = 1, \ldots, NT+1$を対象に定義される$pstock_{pft}$、$rstock_{rft}$ で、期間tの期初に、工場fで保有されている最終製品、および、原材料を表すものとします。

さらに、$MXSELL_{pt}$で、期間tでの、製品pの最大販売可能量を表し、$MXMAKE_f$で、工場fのキャパシティの制限を、*MXRSTOCK*で原材料貯蔵キャパシティを表すものとします。

$IPSTOCK_{pf}$で、計画初期に製品pの、工場fで、貯蔵されている数量を、また、$IRSTOCK_{rf}$で、計画初期に原材料rの、工場fで、貯蔵されている数量を表すものとします。さらに、*CPSTOCK* 、*CRSTOCK* で、それぞれ、最終製品、および、原材料の単位あたりの貯蔵コストを表します。

利益最大の目的関数は、売上(REV_p)から生産コスト($CMAKE_{pf}$)、原材料購入コスト($CBUY_{rt}$)、最終製品、および、原材料の貯蔵コストを差し引いたものです。

$$\mathit{maximize} \sum_{p\in PRODS}\sum_{f\in FACT}\sum_{t\in TIME} REV_{pt}\cdot sell_{pft}$$

$$- \sum_{p\in PRODS}\sum_{f\in FACT}\sum_{t\in TIME} CMAKE_{pt}\cdot make_{pft} - \sum_{r\in RAW}\sum_{f\in FACT}\sum_{t\in TIME} CBUY_{rt}\cdot buy_{rft}$$

$$- \sum_{p\in PRODS}\sum_{f\in FACT}\sum_{t=2}^{NT+1} CPSTOCK\cdot pstock_{pft} - \sum_{r\in RAW}\sum_{f\in FACT}\sum_{t=2}^{NT+1} CRSTOCK\cdot rstock_{rft}$$

期間を跨いで製品を貯蔵することを可能にするには、次の 3 種類の制約式が必要となります。すなわち、最終製品、原材料の*在庫バランス*の制約式(*$PBal_{pft}$*)、および、(*$RBal_{rft}$*)、原材料貯蔵キャパシティの制限の制約式(*$MxRStock_{ft}$*)です。

製品*p*の *t* + 1 期の期初在庫*$pstock_{p,f,t+1}$*は、製品*p*の *t* 期の期初在庫＋*t* 期の生産数量－*t* 期の販売数量です。原材料*r*の *t* + 1 期の期初在庫*$rstock_{r,f,t+1}$*は、原材料*r*の *t* 期の期初在庫＋*t* 期の購入数量－*t* 期の使用数量です。

$$\forall p \in PRODS, \forall f \in FACT, \forall t \in TIME : PBal_{pft} := pstock_{p,f,t+1} = pstock_{pft} + make_{pft} - sell_{pft}$$

$$\forall r \in RAW, \forall f \in FACT, \forall t \in Time :$$
$$RBal_{rft} := rstock_{r,f,t+1} = rstock_{rft} + buy_{rft} - \sum_{p\in PRODS} REQ_{pr}\cdot make_{pft}$$

$$\forall f \in FACT, \forall t \in \{2,...,NT+1\} : MxRStock_{ft} := \sum_{r\in RAW} rstock_{rft} \le MXRSTOCK$$

さらに、2種類のキャパシティ制約式が必要です。すなわち、工場の生産キャパシティは制限されており(制約式 *$MxMake_{ft}$*)、各期間で販売できる最終製品の最大数量は制限されています(制約式*$MxSell_{ft}$*)。

下記は、この問題を数学的に表現したものです。最終製品、および、原材料の*t* = 1期の期初在庫レベルは、与えられた値に固定されています。

$$\mathit{maximize} \sum_{p\in PRODS}\sum_{f\in FACT}\sum_{t\in TIME} REV_{pt}\cdot sell_{pft}$$

$$- \sum_{p\in PRODS}\sum_{f\in FACT}\sum_{t\in TIME} CMAKE_{pt}\cdot make_{pft} - \sum_{r\in RAW}\sum_{f\in FACT}\sum_{t\in TIME} CBUY_{rt}\cdot buy_{rft}$$

$$- \sum_{p\in PRODS}\sum_{f\in FACT}\sum_{t=2}^{NT+1} CPSTOCK\cdot pstock_{pft} - \sum_{r\in RAW}\sum_{f\in FACT}\sum_{t=2}^{NT+1} CRSTOCK\cdot rstock_{rft}$$

$$\forall p \in PRODS, \forall f \in FACT, \forall t \in TIME : PBal_{pft} := pstock_{p,f,t+1} = pstock_{pft} + make_{pft} - sell_{pft}$$

$$\forall r \in RAW, \forall f \in FACT, \forall t \in Time :$$
$$RBal_{rft} := rstock_{r,f,t+1} = rstock_{rft} + buy_{rft} - \sum_{p \in PRODS} REQ_{pr} \cdot make_{pft}$$

$$\forall p \in PRODS, \forall t \in TIME : MxSell_{pt} := \sum_{f \in FACT} sell_{pft} \le MXSELL_{p}$$

$$\forall f \in FACT, \forall t \in TIME : MxMake_{ft} := \sum_{p \in PRODS} make_{pft} \le MXMAKE_{f}$$

$$\forall f \in FACT, \forall t \in \{2,..., NT+1\} : MxRStock_{ft} := \sum_{r \in RAW} rstock_{rft} \le MXRSTOCK$$

$$\forall p \in PRODS, \forall f \in FACT : pstock_{pf1} = IPSTOCK_{pf}$$

$$\forall r \in RAW, \forall f \in FACT : rstock_{rf1} = IRSTOCK_{rf}$$

$$\forall p \in PRODS, \forall f \in FACT, \forall t \in TIME : make_{pft} \ge 0, sell_{pft} \ge 0$$

$$\forall r \in RAW, \forall f \in FACT, \forall t \in TIME : buy_{rft} \ge 0$$

$$\forall p \in PRODS, \forall f \in FACT, \forall t \in \{1,..., NT+1\} : pstock_{pft} \ge 0$$

$$\forall r \in RAW, \forall f \in FACT, \forall t \in \{1,..., NT+1\} : rstock_{rft} \ge 0$$

5.1.2 問題の分解

これから、上で説明したモデルを、生産拠点で分解します。この問題を分解する方法は、ここに示す方法だけではないことに注意してください。この問題は、製品ごとに、または、期間で、分解することもできます。しかし、これらの二つの分解の方法は、工場による分解に比較して、global constraints の数が多くなります。このことは、master problem を解くのが、もっと、難しくなることを意味しています。

工場 *f* ごとに、下記のサブモデルを得ます（boundsの形式での販売数量の制約式*MxSell*をサブモデルにいれることは必須ではありませんが、もっと正確な、もしくは、もっと速いソリューションが得られるかもしれません。）

$$maximize \sum_{p \in PRODS} \sum_{t \in TIME} REV_{pt} \cdot sell_{pt}$$

$$- \sum_{p \in PRODS} \sum_{t \in TIME} CMAKE_p \cdot make_{pt} - \sum_{r \in RAW} \sum_{t \in TIME} CBUY_{rt} \cdot buy_{rt}$$

$$- \sum_{p \in PRODS} \sum_{t=2}^{NT+1} CPSTOCK \cdot pstock_{pt} - \sum_{r \in RAW} \sum_{t=2}^{NT+1} CRSTOCK \cdot rstock_{rt}$$

$$\forall p \in PRODS, \forall t \in TIME : PBal_{pt} := pstock_{p,,t+1} = pstock_{pt} + make_{pt} - sell_{pt}$$

$$\forall r \in RAW, \forall t \in Time : RBal_{rt} := rstock_{r,t+1} = rstock_{rt} + buy_{rt} - \sum_{p \in PRODS} REQ_{pr} \cdot make_{pt}$$

$$\forall t \in TIME : MxMake_t := \sum_{p \in PRODS} make_{pt} \leq MXMAKE$$

$$\forall t \in \{2, ..., NT+1\} : MxRStock_t := \sum_{r \in RAW} rstock_{rt} \leq MXRSTOCK$$

$$\forall p \in PRODS, \forall t \in TIME : MxSell_{pt} := sell_{pt} \leq MXSELL_p$$

$$\forall p \in PRODS, : pstock_{p1} = IPSTOCK_p$$

$$\forall r \in RAW : rstock_{r1} = IRSTOCK_r$$

$$\forall p \in PRODS, \forall t \in TIME : make_{pt} \geq 0, sell_{pt} \geq 0$$

$$\forall r \in RAW, \forall t \in TIME : buy_{rt} \geq 0$$

$$\forall p \in PRODS, \forall t \in \{1, ..., NT+1\} : pstock_{pt} \geq 0$$

$$\forall r \in RAW, \forall t \in \{1, ..., NT+1\} : rstock_{rt} \geq 0$$

master problemは、global constraints、すなわち、販売数量を制限する制約式*MxSell*のみを持っています。（ここでは、変数が非負であることを示す制約式は省いてあります。）

$$maximize \sum_{p \in PRODS} \sum_{f \in FACT} \sum_{t \in TIME} REV_{pt} \cdot sell_{pft}$$

$$- \sum_{p \in PRODS} \sum_{f \in FACT} \sum_{t \in TIME} CMAKE_{pt} \cdot make_{pft} - \sum_{r \in RAW} \sum_{f \in FACT} \sum_{t \in TIME} CBUY_{rt} \cdot buy_{rft}$$

$$- \sum_{p \in PRODS} \sum_{f \in FACT} \sum_{t=2}^{NT+1} CPSTOCK \cdot pstock_{pft} - \sum_{r \in RAW} \sum_{f \in FACT} \sum_{t=2}^{NT+1} CRSTOCK \cdot rstock_{rft}$$

$$\forall p \in PRODS, \forall t \in TIME : MxSell_{pt} := \sum_{f \in FACT} sell_{pft} \leq MXSELL_p$$

分解アルゴリズムでは、master problemのデシジョン変数は、下記のようなサブ問題で生成されるソリューション（proposal)で表現されます。

$$\forall p \in PRODS, \forall f \in FACT, \forall t \in TIME : sell_{pft} = \sum_{k=1}^{nPROP_f} \Pr op\_sell_{pftk} weight_{fk}$$

ここで、$Prop_sell_{pftk}$は、サブ問題fにより生成されたk番目のproposalの中の変数$sell_{pt}$の解の値を意味しています。そして、$Prop_cost_{fk}$は、このproposalのobjective valueです。すべてのサブ問題fにたいして、convexity制約式$Convex_f$を変数$weight_{fk}$に追加する必要があります。

$$\max imize \sum_{f \in FACT} \sum_{k=1}^{nPRop_f} \Pr op\_\cos t_{fk} \cdot weight_{fk}$$

$$\forall p \in PRODS, \forall t \in TIME : MxSell_{pt} \coloneqq \sum_{f \in FACT} \sum_{k=1}^{nPROP_f} \Pr op\_sell_{pftk} \cdot weight_{fk} \le MXSELL_p$$

$$\forall f \in FACT : Convex_f \coloneqq \sum_{k=1}^{nPROP_f} weight_{fk} = 1$$

$$\forall f \in FACT, \forall k \in 1,...,nPROP_f : weight_{fk} \ge 0$$

以下、この問題をmodified master problem と呼びます。これ以上、技術的な詳細に入ることはしませんが、ここでは、もともとの問題とmodified master problem との間には、情報のやりとりがある、ということだけを述べておきます。

5.2 インプリメンテーション

分解アルゴリズムには、下記のフェーズがあります。

- **Phase 1:** modified master problemの実行可能解を得るために、一組のproposalを生成する。
- **Phase 2:** modified master problemを最適化する。
- **Phase 3:**もともとの問題のソリューションを計算する。

フェーズ1と2でかれるサブ問題解は、modified master problem のdual valueの合計を目的関数としてとります。空のmaster problemから出発したり、ミスリーディングなrandom dual informationに影響されないようにしたりするために、フェーズ0として、crashフェーズを追加します。これにより、各サブ問題に、もともとの目的関数を使って、一つのproposalを生成します。

5.2.1 Master model

Below follows the body of the master model file. The definitions of the function bodies will be shown later in Section 5.2.3.

```
model "Coco3 Master"
  uses "mmxprs","mmjobs","mmsystem"

  parameters
    DATAFILE = "coco2.dat"
    ALG = 0 ! 0: stop phase with 1st failed subpb.
                                                  ! 1: stop when all subprob.s fail
  end-parameters

  forward procedure process_sub_result
  forward procedure solve_master(phase:integer)
  forward procedure process_master_result
  forward function calc_solution:real
  forward procedure print_solution

  declarations
    PHASE_0=2                                     ! Event codes sent to submodels
    PHASE_1=3
    PHASE_2=4
    PHASE_3=5
    EVENT_SOLVED=6                                ! Event codes sent by submodels
    EVENT_FAILED=7
    EVENT_READY=8
    NPROD, NFACT, NRAW, NT: integer
  end-declarations

  initializations from DATAFILE
    NPROD NFACT NRAW NT
  end-initializations

  declarations
    PRODS = 1..NPROD                              ! Range of products (p)
    FACT = 1..NFACT                               ! factories (f)
    RAW = 1..NRAW                                 ! raw materials (r)
    TIME = 1..NT                                  ! time periods (t)

    nIter: integer                                ! Iteration counter
    nPROP: array(FACT) of integer                 ! Counters of proposals from subprob.s
  end-declarations

!**** Master problem ****
  declarations
    MXSELL: array(PRODS,TIME) of real             ! Max. amount of p that can be sold
    excessS: mpvar                                ! Violation of sales/buying limits
    weight: array(FACT,range) of mpvar            ! weights for proposals
    MxSell: array(PRODS,TIME) of linctr           ! Sales limit constraints
    Convex: array(FACT) of linctr                 ! Convexity constraints
    Price_convex: array(FACT) of real             ! Dual price on convexity constraints
    Price_sell: array(PRODS,TIME) of real         ! Dual price on sales limits
  end-declarations

  initializations from DATAFILE
    MXSELL
  end-initializations
```

```
!**** Submodels ****
  declarations
    submod: array(FACT) of Model                    ! One subproblem per factory
    Stopped: set of integer
    modid: array(set of integer) of integer         ! Model indices
  end-declarations

  res:= compile("g","cocoSubF.mos")                 ! Compile the submodel file
  forall(f in FACT) do                              ! Load & run one submodel per product
    Price_convex(f):= 1
    load(submod(f), "cocoSubF.bim")
    modid(getid(submod(f))):= f
    run(submod(f), "Factory=" + f + ",DATAFILE=" + DATAFILE)
    wait                                            ! Wait for child model to be ready
    dropnextevent
  end-do

!**** Phase 0: Crash ****
  nIter:=1; finished:=false
  writeln("¥nPHASE 0 -- Iteration ", nIter); fflush
  forall(f in FACT)                                  ! Start solving all submodels (Phase 1)
    send(submod(f), PHASE_0, 0)

  forall(f in FACT) do
  wait                                              ! Wait for child (termination) events
  ev:= getnextevent
  if getclass(ev)=EVENT_SOLVED then
    process_sub_result                              ! Add new proposal to master problem
  elif getclass(ev)=EVENT_FAILED then
    finished:= true
    end-if
  end-do

  if finished then
  writeln("Problem is infeasible")
  exit(1)
  end-if

  solve_master(1)                                   ! Solve the updated Ph. 1 master problem
  process_master_result                             ! Store initial pricing data for submodels

!**** Phase 1: proposal generation (feasibility) ****
  repeat
    noimprove:= 0
    nIter+=1
    writeln("¥nPHASE 1 -- Iteration ", nIter); fflush

    forall(f in FACT)                               ! Start solving all submodels (Phase 1)
      send(submod(f), PHASE_1, Price_convex(f))

    forall(f in FACT) do
    wait                                            ! Wait for child (termination) events
    ev:= getnextevent
    if getclass(ev)=EVENT_SOLVED then
    process_sub_result                              ! Add new proposal to master problem
    elif getclass(ev)=EVENT_FAILED then
      noimprove += 1
    end-if
  end-do

  if noimprove = NFACT then
```

```
      writeln("Problem is infeasible")
      exit(2)
    end-if
    if ALG=0 and noimprove > 0 then
      writeln("No improvement by some subproblem(s)")
      break
    end-if

    solve_master(1)                                  ! Solve the updated Ph. 1 master problem
    if getobjval>0.00001 then
      process_master_result                          ! Store new pricing data for submodels
    end-if
  until getobjval <= 0.00001

  !**** Phase 2: proposal generation (optimization) ****
  writeln("¥n**** PHASE 2 ****")
  finished:=false
  repeat
    solve_master(2)                                  ! Solve Phase 2 master problem
    process_master_result                            ! Store new pricing data for submodels

    nIter+=1
    writeln("¥nPHASE 2 -- Iteration ", nIter); fflush

    forall(f in FACT)                                ! Start solving all submodels (Phase 2)
      send(submod(f), PHASE_2, Price_convex(f))

    forall(f in FACT) do
      wait                                           ! Wait for child (termination) events
      ev:= getnextevent
      if getclass(ev)=EVENT_SOLVED then
        process_sub_result ! Add new proposal to master problem
      elif getclass(ev)=EVENT_FAILED then
        if ALG=0 then
        finished:=true ! 1st submodel w/o prop. stops phase 2
        else
          Stopped += {modid(getfromid(ev))} ! Stop phase 2 only when no submodel
                                                       ! generates a new proposal
        end-if
      end-if
    end-do

    if getsize(Stopped) = NFACT then finished:= true; end-if
  until finished

  solve_master(2)                                    ! Re-solve master to integrate
                                                     ! proposal(s) from last ph. 2 iteration
  !**** Phase 3: solution to the original problem ****
    writeln("¥n**** PHASE 3 ****")
    forall(f in FACT) do
      send(submod(f), PHASE_3, 0)                    ! Stop all submodels
      wait
      dropnextevent
    end-do

    writeln("Total Profit=", calc_solution)
    print_solution
  end-model
```

このモデルの、最初のdeclarations blockにより、このmaster modelとchild (sub)modelとの間

でやりとりされるメッセージを識別するのに使うevent codeが定義されます。child modelでも、同一のdeclarationを行なう必要があります。

5.2.2 単一工場問題

Subproblemを持つモデル・ファイルcocoSubF.mos は、下記の内容を持っています。

```
model "Coco Subproblem (factory based decomp.)"
  uses "mmxprs", "mmjobs"

  parameters
    Factory = 0
    TOL = 0.00001
    DATAFILE = "coco3.dat"
  end-parameters

  forward procedure process_solution

  declarations
    PHASE_0=2                                  ! Event codes sent to submodels
    PHASE_1=3
    PHASE_2=4
    PHASE_3=5
    EVENT_SOLVED=6                             ! Event codes sent by submodels
    EVENT_FAILED=7
    EVENT_READY=8
    NPROD, NFACT, NRAW, NT: integer
  end-declarations

  send(EVENT_READY,0)                          ! Model is ready (= running)

  initializations from DATAFILE
    NPROD NFACT NRAW NT
  end-initializations

  declarations
    PRODS = 1..NPROD                           ! Range of products (p)
    FACT = 1..NFACT                            !        factories (f)
    RAW = 1..NRAW                              !        raw materials (r)
    TIME = 1..NT                               !        time periods (t)

    REV: array(PRODS,TIME) of real             ! Unit selling price of products
    CMAKE: array(PRODS,FACT) of real           ! Unit cost to make product p
                                               ! at factory f
    CBUY: array(RAW,TIME) of real              ! Unit cost to buy raw materials
    REQ: array(PRODS,RAW) of real              ! Requirement by unit of product p
                                               ! for raw material r
    MXSELL: array(PRODS,TIME) of real          ! Max. amount of p that can be sold
    MXMAKE: array(FACT) of real                ! Max. amount factory f can make
                                               ! over all products
    IPSTOCK: array(PRODS,FACT) of real         ! Initial product stock levels
    IRSTOCK: array(RAW,FACT) of real           ! Initial raw material stock levels
    CPSTOCK: real                              ! Unit cost to store any product p
    CRSTOCK: real                              ! Unit cost to store any raw mat. r
    MXRSTOCK: real                             ! Raw material storage capacity
    make: array(PRODS,TIME) of mpvar           ! Amount of products made at factory
    sell: array(PRODS,TIME) of mpvar           ! Amount of product sold from factory
    buy: array(RAW,TIME) of mpvar              ! Amount of raw material bought
    pstock: array(PRODS,1..NT+1) of mpvar      ! Product stock levels at start
                                               ! of period t
```

```
  rstock: array(RAW,1..NT+1) of mpvar                ! Raw material stock levels
                                                     ! at start of period t
  sol_make: array(PRODS,TIME) of real                ! Amount of products made
  sol_sell: array(PRODS,TIME) of real                ! Amount of product sold
  sol_buy: array(RAW,TIME) of real                   ! Amount of raw mat. bought
  sol_pstock: array(PRODS,1..NT+1) of real           ! Product stock levels
  sol_rstock: array(RAW,1..NT+1) of real             ! Raw mat. stock levels
  Profit: linctr                                     ! Profit of proposal
  Price_sell: array(PRODS,TIME) of real              ! Dual price on sales limits
end-declarations

initializations from DATAFILE
  CMAKE REV CBUY REQ MXSELL MXMAKE
  IPSTOCK IRSTOCK MXRSTOCK CPSTOCK CRSTOCK
end-initializations

! Product stock balance
forall(p in PRODS,t in TIME)
  PBal(p,t):= pstock(p,t+1) = pstock(p,t) + make(p,t) - sell(p,t)

! Raw material stock balance
forall(r in RAW,t in TIME)
  RBal(r,t):= rstock(r,t+1) =
    rstock(r,t) + buy(r,t) - sum(p in PRODS) REQ(p,r)*make(p,t)

! Capacity limit
forall(t in TIME)
  MxMake(t):= sum(p in PRODS) make(p,t) <= MXMAKE(Factory)

! Limit on the amount of prod. p to be sold
  forall(p in PRODS,t in TIME) sell(p,t) <= MXSELL(p,t)

! Raw material stock limit
forall(t in 2..NT+1)
  MxRStock(t):= sum(r in RAW) rstock(r,t) <= MXRSTOCK

! Initial product and raw material stock levels
  forall(p in PRODS) pstock(p,1) = IPSTOCK(p,Factory)
  forall(r in RAW) rstock(r,1) = IRSTOCK(r,Factory)

! Total profit
Profit:=
  sum(p in PRODS,t in TIME) REV(p,t) * sell(p,t) -     ! revenue
  sum(p in PRODS,t in TIME) CMAKE(p,Factory) * make(p,t) - ! prod. cost
  sum(r in RAW,t in TIME) CBUY(r,t) * buy(r,t) -       ! raw mat.
  sum(p in PRODS,t in 2..NT+1) CPSTOCK * pstock(p,t) - ! p storage
  sum(r in RAW,t in 2..NT+1) CRSTOCK * rstock(r,t)     ! r storage

! (Re)solve this model until it is stopped by event "PHASE_3"
repeat
  wait
  ev:= getnextevent
  Phase:= getclass(ev)
  if Phase=PHASE_3 then                              ! Stop the execution of this model
  break
end-if
Price_convex:= getvalue(ev)                          ! Get new pricing data

if Phase<>PHASE_0 then
  initializations from "raw:noindex"
    Price_sell as "shmem:Price_sell"
  end-initializations
end-if
```

```
    ! (Re)solve this model
      if Phase=PHASE_0 then
        maximize(Profit)
      elif Phase=PHASE_1 then
        maximize(sum(p in PRODS,t in TIME) Price_sell(p,t)*sell(p,t) + Price_convex)
      else                                                          ! PHASE 2
      maximize(
        Profit - sum(p in PRODS,t in TIME) Price_sell(p,t)*sell(p,t) -
          Price_convex)
      end-if

    writeln("Factory ", Factory, " - Obj: ", getobjval,
            " Profit: ", getsol(Profit), " Price_sell: ",
            getsol(sum(p in PRODS,t in TIME) Price_sell(p,t)*sell(p,t) ),
            " Price_convex: ", Price_convex)
    Fflush

    if getobjval > TOL then                                         ! Solution found: send values to master
      process_solution
    elif getobjval <= TOL then                                      ! Problem is infeasible (Phase 0/1) or
      send(EVENT_FAILED,0)                                           ! no improved solution found (Phase 2)
    else
      send(EVENT_READY,0)
    end-if
  until false

  !-------------------------------------------------------------
  ! Process solution data
  procedure process_solution
    forall(p in PRODS,t in TIME) do
      sol_make(p,t):= getsol(make(p,t))
      sol_sell(p,t):= getsol(sell(p,t))
    end-do
    forall(r in RAW,t in TIME) sol_buy(r,t):= getsol(buy(r,t))
    forall(p in PRODS,t in 1..NT+1) sol_pstock(p,t):= getsol(pstock(p,t))
    forall(r in RAW,t in 1..NT+1) sol_rstock(r,t):= getsol(rstock(r,t))
    Prop_cost:= getsol(Profit)
    send(EVENT_SOLVED,0)

    initializations to "mempipe:noindex,sol"
      Factory
      sol_make sol_sell sol_buy sol_pstock sol_rstock
      Prop_cost
    end-initializations
  end-procedure
  end-model
```

child modelは、PHASE_3 codeを得るまで、何度も、繰り返し解かれます。各々の繰り返しで、child modelはメモリにソリューションを書き、master modelがソリューションを処理できるようにします。

5.2.3 Master problem の subroutines

下記のmaster modelの三つのサブルーチンにより、master modelは、サブルーチン(process_sub_result)が得たソリューションを得て、master問題(solve_master),を繰り返し解き、master modelのソリューションをchild model(process_master_result)に渡します。

```
declarations
  Prop_make: array(PRODS,FACT,TIME,range) of real ! Amount of products made
  Prop_sell: array(PRODS,FACT,TIME,range) of real ! Amount of product sold
  Prop_buy: array(RAW,FACT,TIME,range) of real ! Amount of raw mat. bought
  Prop_pstock: array(PRODS,FACT,1..NT+1,range) of real ! Product stock levels
  Prop_rstock: array(RAW,FACT,1..NT+1,range) of real ! Raw mat. stock levels
  Prop_cost: array(FACT,range) of real ! Cost/profit of each proposal
end-declarations

procedure process_sub_result
  declarations
    f: integer                                  ! Factory index
                                                ! Solution values of the proposal:
    sol_make: array(PRODS,TIME) of real          ! Amount of products made
    sol_sell: array(PRODS,TIME) of real         ! Amount of product sold
    sol_buy: array(RAW,TIME) of real            ! Amount of raw mat. bought
    sol_pstock: array(PRODS,1..NT+1) of real    ! Product stock levels
    sol_rstock: array(RAW,1..NT+1) of real      ! Raw mat. stock levels
    pc: real                                    ! Cost of the proposal
  end-declarations

! Read proposal data from memory
  initializations from "mempipe:noindex,sol"
    f
    sol_make sol_sell sol_buy sol_pstock sol_rstock
    pc
  end-initializations

! Add the new proposal to the master problem
  nPROP(f)+=1
  create(weight(f,nPROP(f)))
  forall(p in PRODS,t in TIME) do
    Prop_make(p,f,t,nPROP(f)):= sol_make(p,t)
    Prop_sell(p,f,t,nPROP(f)):= sol_sell(p,t)
  end-do
  forall(r in RAW,t in TIME) Prop_buy(r,f,t,nPROP(f)):= sol_buy(r,t)
  forall(p in PRODS,t in 1..NT+1) Prop_pstock(p,f,t,nPROP(f)):= sol_pstock(p,t)
  forall(r in RAW,t in 1..NT+1) Prop_rstock(r,f,t,nPROP(f)):= sol_rstock(r,t)
  Prop_cost(f,nPROP(f)):= pc
  writeln("Sol. for factory ", f, ":¥n make: ", sol_make, "¥n sell: ",
          sol_sell, "¥n buy: ", sol_buy, "¥n pstock: ", sol_pstock,
  "¥n rstock: ", sol_rstock)
end-procedure

!-------------------------------------------------------------
  procedure solve_master(phase: integer)
    forall(f in FACT)
      Convex(f):= sum (k in 1..nPROP(f)) weight(f,k) = 1
    if phase=1 then
      forall(p in PRODS,t in TIME)
        MxSell(p,t):=
          sum(f in FACT,k in 1..nPROP(f)) Prop_sell(p,f,t,k)*weight(f,k) -
            excessS <= MXSELL(p,t)
    minimize(excessS)
  else
    forall(p in PRODS,t in TIME)
      MxSell(p,t):=
        sum(f in FACT,k in 1..nPROP(f)) Prop_sell(p,f,t,k)*weight(f,k) <=
          MXSELL(p,t)
    maximize(sum(f in FACT, k in 1..nPROP(f)) Prop_cost(f,k) * weight(f,k))
  end-if
```

```
    writeln("Master problem objective: ", getobjval)
    write(" Weights:")
    forall(f in FACT,k in 1..nPROP(f)) write(" ", getsol(weight(f,k)))
    writeln
end-procedure

!-------------------------------------------------------------
procedure process_master_result
    forall(p in PRODS,t in TIME) Price_sell(p,t):=getdual(MxSell(p,t))
    forall(f in FACT) Price_convex(f):=getdual(Convex(f))

initializations to "raw:noindex"
    Price_sell as "shmem:Price_sell"
end-initializations
end-procedure
```

最後に、master modelは、もともとの問題(calc_solution)へのソリューションを計算する二つのサブルーチンにより完了し（decomposition algorithmのフェーズ3）、ソリューションがプリントされます(print_solution)。もともとの問題のソリューションは、modified master problemの値とsubproblemで生成されるproposalとから得ます。

```
declarations
    REV: array(PRODS,TIME) of real              ! Unit selling price of products
    CMAKE: array(PRODS,FACT) of real            ! Unit cost to make product p
                                                ! at factory f
    CBUY: array(RAW,TIME) of real               ! Unit cost to buy raw materials
    COPEN: array(FACT) of real                  ! Fixed cost of factory f being
                                                ! open for one period
    CPSTOCK: real                               ! Unit cost to store any product p
    CRSTOCK: real                               ! Unit cost to store any raw mat. R

    Sol_make: array(PRODS,FACT,TIME) of real ! Solution value (products made)
    Sol_sell: array(PRODS,FACT,TIME) of real ! Solution value (product sold)
    Sol_buy: array(RAW,FACT,TIME) of real ! Solution value (raw mat. bought)
    Sol_pstock: array(PRODS,FACT,1..NT+1) of real ! Sol. value (prod. stock)
    Sol_rstock: array(RAW,FACT,1..NT+1) of real ! Sol. value (raw mat. stock)
end-declarations

initializations from DATAFILE
    CMAKE REV CBUY CPSTOCK CRSTOCK COPEN
end-initializations

function calc_solution: real
    forall(p in PRODS,f in FACT,t in TIME) do
        Sol_sell(p,f,t):=
            sum(k in 1..nPROP(f)) Prop_sell(p,f,t,k) * getsol(weight(f,k))
        Sol_make(p,f,t):=
            sum(k in 1..nPROP(f)) Prop_make(p,f,t,k) * getsol(weight(f,k))
end-do
forall(r in RAW,f in FACT,t in TIME) Sol_buy(r,f,t):=
        sum(k in 1..nPROP(f)) Prop_buy(r,f,t,k) * getsol(weight(f,k))
forall(p in PRODS,f in FACT,t in 1..NT+1) Sol_pstock(p,f,t):=
        sum(k in 1..nPROP(f)) Prop_pstock(p,f,t,k) * getsol(weight(f,k))
forall(r in RAW,f in FACT,t in 1..NT+1) Sol_rstock(r,f,t):=
        sum(k in 1..nPROP(f)) Prop_rstock(r,f,t,k) * getsol(weight(f,k))

returned:=
        sum(p in PRODS,f in FACT,t in TIME) REV(p,t) * Sol_sell(p,f,t) -
```

```
      sum(p in PRODS,f in FACT,t in TIME) CMAKE(p,f) * Sol_make(p,f,t) -
      sum(r in RAW,f in FACT,t in TIME) CBUY(r,t) * Sol_buy(r,f,t) -
      sum(p in PRODS,f in FACT,t in 2..NT+1) CPSTOCK * Sol_pstock(p,f,t) -
      sum(r in RAW,f in FACT,t in 2..NT+1) CRSTOCK * Sol_rstock(r,f,t)
end-function

!-------------------------------------------------------------
procedure print_solution
  writeln("Finished products:")
  forall(f in FACT) do
    writeln("Factory ", f, ":")
    forall(p in PRODS) do
    write(" ", p, ": ")
    forall(t in TIME) write(strfmt(Sol_make(p,f,t),6,1), "(",
                            strfmt(Sol_pstock(p,f,t+1),5,1), ")")
    writeln
  end-do
end-do

writeln("Raw material:")
forall(f in FACT) do
  writeln("Factory ", f, ":")
  forall(r in RAW) do
    write(" ", r, ": ")
    forall(t in TIME) write(strfmt(Sol_buy(r,f,t),6,1), "(",
                            strfmt(Sol_rstock(r,f,t+1),5,1), ")")
    writeln
  end-do
end-do

writeln("Sales:")
forall(f in FACT) do
    writeln("Factory ", f, ":")
    forall(p in PRODS) do
    write(" ", p, ": ")
    forall(t in TIME) write(strfmt(Sol_sell(p,f,t),4))
    writeln
  end-do
end-do

writeln("¥nComputation time: ", gettime)
end-procedure
```

5.3 結果

ここでのテスト・ケースに関しては、decomposition algorithm を使って得られたソリューションは、最適解に近いものですが、最適解は、必ずしも得られるとは限りません。その理由は、decomposition algorithm が繰り返しの連続で、これにより、いろいろなレベルでエラーが蓄積し、submodelでの停止判定基準（stopping criterion ）として使われるトレランスを低め、多くの場合、解を改善しなくなるからです。しかし、decomposition algorithm のコンフィギュレーションそれ自体により、解に影響することもできます。例えば､フェーズ1、フェーズ2で、最初のsubmodel が改善された解を返して寄越さなくなったときに停止する、つまり、新しいproposalが生成されなくなるまで続けるという方針です｡たくさんのproposalを生成することで、より良いソリューションが見つかるかもしれませんが、これにより、(sub)problemsを解く回数も増加し、したがって、解く時間が長くなります。

6 Benders decomposition: working with several different submodels

Benders decompositionは、大きなMIP問題を解くためのdecomposition methodの一つです。 標準的な解法を使って解くには大きすぎるMIP問題を、そのまま解く代わりに、一連の線形、および、pure integer subproblemsを使いますが、このとき、後者の制約式の数は、もともとの問題のそれよりも少なくなります。下記のdecomposition algorithmについての説明は、[Hu69]から取ったものです。ここには、この方法の有限性（finiteness）、および、この方法が、常に、最適解をもたらすことの証明がありますので、興味のある読者はご覧ください。

下記のMIP問題(*problem I*)の標準形式について考えましょう。

$$\text{minimize } z = \mathbf{Cx} + \mathbf{Dy}$$
$$\mathbf{Ax} + \mathbf{By} \geq \mathbf{b}$$
$$\mathbf{x}, \mathbf{y} \geq \mathbf{0}, \mathbf{y} \text{ integer}$$

上の定式化、および、本章を通じて、ベクトル、および、マトリックスには太字を使います。例えば、$\mathbf{Cx} + \mathbf{Dy}$ は、$\sum_{i=1}^{NCTVAR} C_i \cdot x_i + \sum_{i=1}^{NINTVAR} D_i \cdot y_i$ を意味し、ここで、*NCTVAR*は連続変数の数を表し、また、*NINTVAR*はインテジャー変数の数を表します。yが特定の値$\mathbf{y}'$を取るとき、上の問題は線形問題になり、下記のように表現できます(*problem II*)。

$$\text{minimize } \mathbf{Cx}$$
$$\mathbf{Ax} \geq \mathbf{b} - \mathbf{By}'$$
$$\mathbf{x} \geq \mathbf{0}$$

*problem II*の双対問題は、下記の*problem IId*のようになります。

$$\text{maximize } \mathbf{u}(\mathbf{b} - \mathbf{By}')$$
$$\mathbf{uA} \leq \mathbf{C}$$
$$\mathbf{u} \geq \mathbf{0}$$

双対問題*IId* の興味深い特徴は、uの実行可能領域が、yに独立であるということです。さらに、双対定理から、もし、問題*IId* がインフィージブルであったり、有限最適解を持たなかったりするならば、もともとの問題*problem I*も有限最適解を持たない、ということがいえます。また、双対定理から、問題*IId* が有限最適解$\mathbf{u}^*$を持つならば、このソリューションは、主問題(すなわち、$\mathbf{u}^* (\mathbf{b} - \mathbf{By}') = \mathbf{Cx}^*$)の最適解と同じ値を持ち、実行可能領域の頂点pでの解$\mathbf{u}^p$は、$\mathbf{u}^p(\mathbf{b} - \mathbf{By}') \leq \mathbf{Cx}^*$という値を持つということがいえます。もともとのMIP問題の目的関数に、実行可能領域の頂点pでの解$\mathbf{u}^p$ を代入すると、下記のような制約式を得ます。

$$z \geq \mathbf{u}^p (\mathbf{b} - \mathbf{By}) + \mathbf{Dy}$$

もともとのMIP問題(*problem I*)の最適解$\mathbf{z}^*$を得るには、*Benders decomposition algorithm*として知られている、下記のpartitioning algorithm を使うことができます。

Step 0 Find a $\mathbf{u}'$ that satisfies $\mathbf{u}'\mathbf{A} \leq \mathbf{C}$

if no such $\mathbf{u}'$ exists

then the original problem (I) has no feasible solution

else continue with Step 1

end-if

Step 1 **Solve the pure integer program**

$$\text{minimize } \mathbf{z}$$

$$\mathbf{z} \geq \mathbf{u}'(\mathbf{b} - \mathbf{By}) + \mathbf{Dy}$$

$$\mathbf{y} \geq \mathbf{0}, \mathbf{y} \text{ integer}$$

If z is unbounded from below, take a z to be any small value $\mathbf{z}'$.

Step 2 **With the solution $\mathbf{y}'$ of Step 1, solve the linear program**

$$\text{maximize } \mathbf{u}(\mathbf{b} - \mathbf{By}')$$

$$\mathbf{uA} \leq \mathbf{C}$$

$$\mathbf{u} \geq \mathbf{0}$$

If u goes to inifinity with $\mathbf{u}(\mathbf{b} - \mathbf{By}')$

then add the constraintPi $\sum_i u_i \leq M$**, where *M* is a large positive constant, and resolve the problem**

end-if

Let the solution of this program be $\mathbf{u}''$.

if $\mathbf{z}' - \mathbf{Dy}' \leq \mathbf{u}''(\mathbf{b} - \mathbf{By}')$

then continue with Step 3

else return to Step 1 and add the constraint $\mathbf{z}' \geq \mathbf{Dy} + \mathbf{u}''(\mathbf{b} - \mathbf{By})$

end-if

Step 3 **With the solution $\mathbf{y}'$ of Step 1, solve the linear program**

$$\text{minimize } \mathbf{Cx}$$

$$\mathbf{Ax} \geq \mathbf{b} - \mathbf{By}'$$

$$\mathbf{x} \geq \mathbf{0}$$

$\mathbf{x}'$ and $\mathbf{y}'$ are the optimum solution $\mathbf{z}^* = \mathbf{Cx}' + \mathbf{Dy}'$

このアルゴリズムは、有限の解をもたらすことが証明できます（provably finite）。そして、最適解をもたらし、実行の中のどの時点でも、最適解z^*の上限、下限がわかります。

6.1 小さい例題

ここで、Benders decompositionを使って、下記の小さい問題を解いてみましょう。この問題の連続変数x_iの数は*NCTVAR* = 3、インテジャー変数y_iの数は*NINTVAR* = 3です。また、不等式の制約式は*NC* = 4です。

$$\begin{aligned}
\text{maximize } & 5\cdot x_1 + 4\cdot x_2 + 3\cdot x_3 + 2\cdot y_1 + 2\cdot y_2 + 3\cdot y_3 \\
& x_1 - x_2 + 5\cdot x_3 + 3\cdot y_1 + 2\cdot y_3 \le 5 \\
& 4\cdot x_2 + 3\cdot x_3 - y_1 + y_2 \le 7 \\
& 2\cdot x_1 - 2\cdot x_3 + y_1 - y_2 + y_3 \le 4 \\
& 3\cdot x_1 + 5\cdot y_1 + 5\cdot y_2 + 5\cdot y_3 \le -2 \\
& \mathbf{x}, \mathbf{y} \ge \mathbf{0}, \mathbf{y} \text{ integer}
\end{aligned}$$

6.2 インプリメンテーション

6.2.1 Master model

master modelはデータを読み、ソリューション・アルゴリズムを定義し、サブモデルとのやり取りを調整し、最後にソリューションをプリントします。（fixed integer variables を使って双対問題を解く)アルゴリズムのステップ2では、主問題を解き、双対問題の解の値をOptimizer から得るか、もしくは、自分の手で双対問題を定義して、それを解くか、という選択ができます。Model parameter ALGにより、ユーザは、どちらを採るかを選択します。

master modelの主要部分は、下記のようなものになります。ソリューション・アルゴリズムをスタートさせる前に、サブモデルを、すべて、コンパイルし、ロードし、スタートします。それは、アルゴリズムのそれぞれのステップで、対応するサブモデルの(re)solvingをトリガーするだけで済むようにするためです。

```
model "Benders (master model)"
  uses "mmxprs", "mmjobs"

  parameters
    NCTVAR = 3
    NINTVAR = 3
    NC = 4
    BIGM = 1000
    ALG = 1                                  ! 1: Use Benders decomposition (dual)
                                             ! 2: Use Benders decomposition (primal)
    DATAFILE = "bprob33.dat"
  end-parameters

  forward procedure start_solution
  forward procedure solve_primal_int(ct: integer)
  forward procedure solve_cont
```

```
  forward function eval_solution: boolean
  forward procedure print_solution

  declarations
    STEP_0=2                              ! Event codes sent to submodels
    STEP_1=3
    STEP_2=4
    EVENT_SOLVED=6                        ! Event codes sent by submodels
    EVENT_INFEAS=7
    EVENT_READY=8

    CtVars = 1..NCTVAR                    ! Continuous variables
    IntVars = 1..NINTVAR                  ! Discrete variables
    Ctrs = 1..NC                          ! Set of constraints (orig. problem)

    A: array(Ctrs,CtVars) of integer      ! Coeff.s of continuous variables
    B: array(Ctrs,IntVars) of integer     ! Coeff.s of discrete variables
    b: array(Ctrs) of integer             ! RHS values
    C: array(CtVars) of integer           ! Obj. coeff.s of continuous variables
    D: array(IntVars) of integer          ! Obj. coeff.s of discrete variables
    Ctr: array(Ctrs) of linctr            ! Constraints of orig. problem
    CtrD: array(CtVars) of linctr         ! Constraints of dual problem
    MC: array(range) of linctr            ! Constraints generated by alg.

    sol_u: array(Ctrs) of real            ! Solution of dual problem
    sol_x: array(CtVars) of real          ! Solution of primal prob. (cont.)
    sol_y: array(IntVars) of real         ! Solution of primal prob. (integers)
    sol_obj: real                         ! Objective function value (primal)
    RM: range                             ! Model indices
    stepmod: array(RM) of Model           ! Submodels
  end-declarations

initializations from DATAFILE
  A B b C D
end-initializations

! **** Submodels ****

initializations to "raw:noindex"          ! Save data for submodels
  A as "shmem:A" B as "shmem:B"
  b as "shmem:b" C as "shmem:C" D as "shmem:D"
end-initializations

! Compile + load all submodels
if compile("benders_int.mos")<>0 then exit(1); end-if
create(stepmod(1)); load(stepmod(1), "benders_int.bim")
if compile("benders_dual.mos")<>0 then exit(2); end-if

if ALG=1 then
  create(stepmod(2)); load(stepmod(2), "benders_dual.bim")
else
  create(stepmod(0)); load(stepmod(0), "benders_dual.bim")
  if compile("benders_cont.mos")<>0 then exit(3); end-if
  create(stepmod(2)); load(stepmod(2), "benders_cont.bim")
  run(stepmod(0), "NCTVAR=" + NCTVAR + ",NINTVAR=" + NINTVAR + ",NC=" + NC)
end-if

                                          ! Start the execution of the submodels
run(stepmod(1), "NINTVAR=" + NINTVAR + ",NC=" + NC)
run(stepmod(2), "NCTVAR=" + NCTVAR + ",NINTVAR=" + NINTVAR + ",NC=" + NC)

forall(m in RM) do
  wait                                    ! Wait for "Ready" messages
```

```
   ev:= getnextevent
   if getclass(ev) <> EVENT_READY then
      writeln("Error occurred in a subproblem")
      exit(4)
end-if
end-do

! **** Solution algorithm ****

start_solution                          ! 0. Initial solution for cont. var.s
ct:= 1
repeat
writeln("¥n**** Iteration: ", ct)
   solve_primal_int(ct)                 ! 1. Solve problem with fixed cont.
   solve_cont                           ! 2. Solve problem with fixed int.
   ct+=1
until eval_solution                     !    Test for optimality
print_solution                          ! 3. Retrieve and display the solution
```

異なるサブモデルを開始したサブルーチンは、'start solving' eventを送信し、サブモデルが解けたら、ソリューションをリトリーブします。start solution の生成には、パラメータALG のセッティングに従い、正しいサブモデルを選択する必要があります。もしこの問題がインフィージブルであることが判明したら、問題全体がインフィージブルであるので、モデルの実行を停止します。

```
! Produce an initial solution for the dual problem using a dummy objective
   procedure start_solution
      if ALG=1 then                     ! Start the problem solving
         send(stepmod(2), STEP_0, 0)
      else
         send(stepmod(0), STEP_0, 0)
      end-if
      wait                              ! Wait for the solution
      ev:=getnextevent
      if getclass(ev)=EVENT_INFEAS then
         writeln("Problem is infeasible")
         exit(6)
      end-if
end-procedure

!-------------------------------------------------------------
! Solve a modified version of the primal problem, replacing continuous
! variables by the solution of the dual
   procedure solve_primal_int(ct: integer)
      send(stepmod(1), STEP_1, ct)      ! Start the problem solving
      wait                              ! Wait for the solution
      ev:=getnextevent
      sol_obj:= getvalue(ev)            ! Store objective function value

   initializations from "raw:noindex"   ! Retrieve the solution
      sol_y as "shmem:y"
   end-initializations
end-procedure

!-------------------------------------------------------------
! Solve the Step 2 problem (dual or primal depending on value of ALG)
! for given solution values of y
   procedure solve_cont
      send(stepmod(2), STEP_2, 0)       ! Start the problem solving
```

```
    wait                                        ! Wait for the solution
    dropnextevent

  initializations from "raw:noindex"            ! Retrieve the solution
    sol_u as "shmem:u"
  end-initializations
end-procedure
```

master modelは、termination criterion (function eval_- solution)が満たされたかどうかをテストし、最終ソリューションをプリントします(procedure print_solution)。この手順は、また、すべてのサブモデルを停止します。

```
function eval_solution: boolean
  write("Test optimality: ", sol_obj - sum(i in IntVars) D(i)*sol_y(i),
          " = ", sum(j in Ctrs) sol_u(j)* (b(j) -
                                sum(i in IntVars) B(j,i)*sol_y(i)) )
  returned:= ( sol_obj - sum(i in IntVars) D(i)*sol_y(i) =
      sum(j in Ctrs) sol_u(j)* (b(j) - sum(i in IntVars) B(j,i)*sol_y(i)) )
  writeln(if(returned, " : true", " : false"))
end-function

!-------------------------------------------------------------
procedure print_solution
  ! Retrieve results
  initializations from "raw:noindex"
    sol_x as "shmem:x"
  end-initializations

  forall(m in RM) stop(stepmod(m))                  ! Stop all submodels

  write("¥n**** Solution (Benders): ", sol_obj, "¥n x: ")
  forall(i in CtVars) write(sol_x(i), " ")
  write(" y: ")
  forall(i in IntVars) write(sol_y(i), " ")
  writeln
end-procedure
```

6.2.2 Submodel 1: fixed continuous variables

decomposition algorithmの最初のステップで、pure integer problemを解く必要があります。このモデルの実行が開始されるとき、モデルはinvariant data を読み込み、変数をセットアップします。次いで、wait statement (first line of the repeat-until loop) により、master modelが(solving) eventを送ってくるまで休止します。この問題を解く度に、メモリから読んでくる連続問題の、前回のランの解の値を使って、integer problemの新しい制約式MC(k)が定義されます。モデル全体、および、解法が埋め込まれているendless loop は、master modelからの‘stop model’commandによってのみ、終了されます。このサブモデル(file benders_- int.mos)のソース全体を下記にリストします。

```
  model "Benders (integer problem)"
    uses "mmxprs", "mmjobs"

    parameters
```

```
    NINTVAR = 3
    NC = 4
    BIGM = 1000
   end-parameters

   declarations
    STEP_0=2                              ! Event codes sent to submodels
    STEP_1=3
    EVENT_SOLVED=6                        ! Event codes sent by submodels
    EVENT_READY=8

    IntVars = 1..NINTVAR                  ! Discrete variables
    Ctrs = 1..NC                          ! Set of constraints (orig. problem)

    B: array(Ctrs,IntVars) of integer     ! Coeff.s of discrete variables
    b: array(Ctrs) of integer             ! RHS values
    D: array(IntVars) of integer          ! Obj. coeff.s of discrete variables
    MC: array(range) of linctr            ! Constraints generated by alg.

    sol_u: array(Ctrs) of real            ! Solution of dual problem
    sol_y: array(IntVars) of real         ! Solution of primal prob.

    y: array(IntVars) of mpvar            ! Discrete variables
    z: mpvar                              ! Objective function variable
end-declarations

initializations from "raw:noindex"
  B as "shmem:B" b as "shmem:b" D as "shmem:D"
end-initializations

z is_free                                 ! Objective is a free variable
forall(i in IntVars) y(i) is_integer      ! Integrality condition
forall(i in IntVars) y(i) <= BIGM         ! Set (large) upper bound

send(EVENT_READY,0)                       ! Model is ready (= running)

repeat
  wait
  ev:= getnextevent
  ct:= integer(getvalue(ev))

initializations from "raw:noindex"
  sol_u as "shmem:u"
end-initializations

! Add a new constraint
MC(ct):= z >= sum(i in IntVars) D(i)*y(i) +
                sum(j in Ctrs) sol_u(j)*(b(j) - sum(i in IntVars) B(j,i)*y(i))
minimize(z)

! Store solution values of y
forall(i in IntVars) sol_y(i):= getsol(y(i))

initializations to "raw:noindex"
  sol_y as "shmem:y"
end-initializations

send(EVENT_SOLVED, getobjval)

write("Step 1: ", getobjval, "¥n y: ")
forall(i in IntVars) write(sol_y(i), " ")

write("¥n Slack: ")
```

```
    forall(j in 1..ct) write(getslack(MC(j)), " ")
    writeln
    fflush
    until false
  end-model
```

我われが解いているこの問題は、イタレーションの次の問題とたった一つの制約式が違うだけですから、root LP-緩和のソリューションのベイシス(MIP solutionのベイシスではない)をセーブして置き、次の最適化のランのときにリロードすることは行う価値があります。しかし、ここでの小さなテスト用のモデルでは、実行スピードの改善にあまり役立ちません。ベイシスをセーブし、読み込むには、call to minimizeを、下記の一連のステートメントで置き換えます。

```
declarations
  bas: basis
end-declarations

loadprob(z)
loadbasis(bas)
minimize(XPRS_TOP, z)
savebasis(bas)
minimize(XPRS_GLB, z)
```

6.2.3 Submodel 2: fixed integer variables

decomposition algorithm の二番目のステップは、すべてのインテジャー変数を、一番目のステップで見つかったソリューションの値に固定されたサブモデルを解くことです。このステップを実行するモデルの構造は、前のサブモデルと非常によく似ています。モデルがランされると、モデルは、メモリからinvariant dataを読み、目的関数をセットアップします。次いで、モデルは、ループの最初で、line waitで休止し、master modelから、step 2 solving event が送られてくるのを待ちます。解の反復のたびごとに、メモリから読み込まれる係数を使って、constraints CTR は再定義され、ソリューションは、メモリ返されます。下記は、このモデルのソースです(file benders_cont.mos)。

```
model "Benders (continuous problem)"
  uses "mmxprs", "mmjobs"

parameters
  NCTVAR = 3
  NINTVAR = 3
  NC = 4
  BIGM = 1000
end-parameters

declarations
  STEP_0=2                              ! Event codes sent to submodels
  STEP_2=4
  STEP_3=5
  EVENT_SOLVED=6                        ! Event codes sent by submodels
  EVENT_READY=8

  CtVars = 1..NCTVAR                    ! Continuous variables
```

```
  IntVars = 1..NINTVAR                    ! Discrete variables
  Ctrs = 1..NC                            ! Set of constraints (orig. problem)

  A: array(Ctrs,CtVars) of integer        ! Coeff.s of continuous variables
  B: array(Ctrs,IntVars) of integer       ! Coeff.s of discrete variables
  b: array(Ctrs) of integer               ! RHS values
  C: array(CtVars) of integer             ! Obj. coeff.s of continuous variables
  Ctr: array(Ctrs) of linctr              ! Constraints of orig. problem

  sol_u: array(Ctrs) of real              ! Solution of dual problem
  sol_x: array(CtVars) of real            ! Solution of primal prob. (cont.)
  sol_y: array(IntVars) of real           ! Solution of primal prob. (integers)

  x: array(CtVars) of mpvar               ! Continuous variables
end-declarations

initializations from "raw:noindex"
  A as "shmem:A" B as "shmem:B"
  b as "shmem:b" C as "shmem:C"
end-initializations

Obj:= sum(i in CtVars) C(i)*x(i)

send(EVENT_READY,0)                       ! Model is ready (= running)

! (Re)solve this model until it is stopped by event "STEP_3"
  repeat
    wait
    dropnextevent

    initializations from "raw:noindex"
      sol_y as "shmem:y"
    end-initializations

    forall(j in Ctrs)
      Ctr(j):= sum(i in CtVars) A(j,i)*x(i) +
                sum(i in IntVars) B(j,i)*sol_y(i) >= b(j)

minimize(Obj)                             ! Solve the problem

! Store values of u and x
  forall（j in Ctrs) sol_u(j):= getdual(Ctr(j))
  forall(i in CtVars) sol_x(i):= getsol(x(i))

  initializations to "raw:noindex"
    sol_u as "shmem:u" sol_x as "shmem:x"
  end-initializations

  send(EVENT_SOLVED, getobjval)

  write("Step 2: ", getobjval, "¥n u: ")
  forall(j in Ctrs) write(sol_u(j), " ")
  write("¥n x: ")
  forall(i in CtVars) write(getsol(x(i)), " ")
  writeln
  fflush

until false

end-model
```

6.2.4 Submodel 0: start solution

decomposition algorithm を開始するには、連続変数に、初期値を生成する必要があります。これは、ダミー目的関数を使い、双対問題を解くことで行ないます。このアルゴリズムのステップ2でも、双対問題を使いますが、これは、前の章で見た主問題を置き換えるときです。このサブモデルのインプリメンテーションでは、二つのケースを考えます。solving loop の中で、master modelにより送られたtype (class) of eventをテストし、それによって解くべき問題を選択します。

このモデルの主要部分は、下記のMosel code (file benders_- dual.mos)で実行されます。

```
model "Benders (dual problem)"
 uses "mmxprs", "mmjobs"

 parameters
  NCTVAR = 3
  NINTVAR = 3
  NC = 4
  BIGM = 1000
 end-parameters

 forward procedure save_solution

 declarations
  STEP_0=2                                  ! Event codes sent to submodels
  STEP_2=4
  EVENT_SOLVED=6                            ! Event codes sent by submodels
  EVENT_INFEAS=7
  EVENT_READY=8

  CtVars = 1..NCTVAR                        ! Continuous variables
  IntVars = 1..NINTVAR                      ! Discrete variables
  Ctrs = 1..NC                              ! Set of constraints (orig. problem)

  A: array(Ctrs,CtVars) of integer          ! Coeff.s of continuous variables
  B: array(Ctrs,IntVars) of integer         ! Coeff.s of discrete variables
  b: array(Ctrs) of integer                 ! RHS values
  C: array(CtVars) of integer               ! Obj. coeff.s of continuous variables

  sol_u: array(Ctrs) of real                ! Solution of dual problem
  sol_x: array(CtVars) of real              ! Solution of primal prob. (cont.)
  sol_y: array(IntVars) of real             ! Solution of primal prob. (integers)

  u: array(Ctrs) of mpvar                   ! Dual variables
 end-declarations

 initializations from "raw:noindex"
  A as "shmem:A" B as "shmem:B"
  b as "shmem:b" C as "shmem:C"
 end-initializations

 forall(i in CtVars) CtrD(i):= sum(j in Ctrs) u(j)*A(j,i) <= C(i)

 send(EVENT_READY,0)                        ! Model is ready (= running)

! (Re)solve this model until it is stopped by event "STEP_3"
 repeat
  wait
```

```
    ev:= getnextevent
    Alg:= getclass(ev)

    if Alg=STEP_0 then                          ! Produce an initial solution for the
                                                ! dual problem using a dummy objective
    maximize(sum(j in Ctrs) u(j))
    if(getprobstat = XPRS_INF) then
      writeln("Problem is infeasible")
      send(EVENT_INFEAS,0)                      ! Problem is infeasible
    else
      write("**** Start solution: ")
      save_solution
   end-if

  else                                          ! STEP 2: Solve the dual problem for
                                                ! given solution values of y
   initializations from "raw:noindex"
    sol_y as "shmem:y"
   end-initializations

   Obj:= sum(j in Ctrs) u(j)* (b(j) - sum(i in IntVars) B(j,i)*sol_y(i))
   maximize(XPRS_PRI, Obj)

   if(getprobstat=XPRS_UNB) then
    BigM:= sum(j in Ctrs) u(j) <= BIGM
    maximize(XPRS_PRI, Obj)
   end-if

   write("Step 2: ")
   save_solution                                ! Write solution to memory
   BigM:= 0                                     ! Reset the 'BigM' constraint
  end-if

  until false
```

このモデルは、subroutine save_solutionで終わります。subroutine save_solutionは、メモリにソリューションを書き、EVENT_- SOLVED message を送ることで、master modelにソリューションが利用可能であることを知らせます。

```
! Process solution data
 procedure save_solution
   ! Store values of u and x
    forall(j in Ctrs) sol_u(j):= getsol(u(j))
    forall(i in CtVars) sol_x(i):= getdual(CtrD(i))

    initializations to "raw:noindex"
     sol_u as "shmem:u" sol_x as "shmem:x"
    end-initializations

    send(EVENT_SOLVED, getobjval)

    write(getobjval, "¥n u: ")
    forall(j in Ctrs) write(sol_u(j), " ")
    write("¥n x: ")
    forall(i in CtVars) write(getdual(CtrD(i)), " ")
    writeln
    fflush
 end-procedure
end-model
```

6.3 Results

この小さなテストモデルの最適解では、目的関数の値は18.1852です。ここで使ったプログラムにより、下記のアウトプットが得られ、そこから、この問題は、decomposition algorithmを3回繰り返して得られたことが判ります(looping around steps 1 and 2)。

```
**** Start solution: 4.05556
      u: 0.740741    1.18519          2.12963 0
      x: 0.611111    0.166667         0.111111

**** Iteration: 1
Step 1: -1146.15
   y: 1000 0 0
   Slack: 0
Step 2: 1007
   u: 0 1 0 0
   x: 0 251.75 0
Test optimality: -3146.15 = 1007 : false

**** Iteration: 2
Step 1: 17.0185
   y: 3 0 0
   Slack: 0 -1.01852
Step 2: 12.5
   u: 0 1 2.5 0
   x: 0.5 2.5 0
Test optimality: 11.0185 = 12.5 : false

**** Iteration: 3
Step 1: 18.1852
   y: 2 0 0
   Slack: 0 -5.18519 -0.185185
Step 2: 14.1852
   u: 0.740741 1.18519 2.12963 0
   x: 1.03704 2.22222 0.037037
Test optimality: 14.1852 = 14.1852 : true


**** Solution (Benders): 18.1852
      x: 1.03704      2.22222          0.037037       y: 2 0 0
```

７要約

このホワイトペーパーの中で示されている例は、*mmjobs* モジュールで用意した機能をどのように使うかを示しております。これらの例は、読者の皆さんが、技術的な可能性を網羅的に概観することなしに、問題をパラレルに解いたり、その他の複数問題解法アプローチを実際に使ったり、試してみたりするためのスターティングポイントとするためのものです。

Mosel で使えるどの solver モジュールも *mmjobs*.と連動して使えます。ただし、同一の solver で複数問題をパラレルに解くことは、使おうとする solver が multi-threaded 環境で動く場合においてのみ可能です。Xpress-Optimizer、それから派生した QP、SLP、SP、および、Xpress-Kalis は、この条件を満たしています。

Bibliography

[Hu69] T. C. Hu. *Integer programming and network flows*. Addison-Wesley, Reading, MA, 1969.

[PW94] Y. Pochet and L. A. Wolsey. Algorithms and Reformulations for Lot Sizing Problems. Technical Report Discussion Paper 9427, CORE, Université Catholique de Louvain, 1994.

[Teb01] J. R. Tebboth. *A computational study of Dantzig-Wolfe decomposition*. PhD thesis, University of Buckingham, 2001.