dash optimization

# Hybrid MIP/CP solving

## with Xpress-Optimizer and Xpress-Kalis

## Dash Optimization Whitepaper

Last update 23 November, 2005

[これは、皆様のご便宜のための参考翻訳です。
誤訳、不適切な訳があるかもしれませんので、
原文と併読なさるようにしてください。]

Published by Dash Optimization Ltd

# How to Contact Dash

## USA, Canada and all Americas

Dash Optimization Inc
Information and Sales: info@dashoptimization.com
Licensing: license-usa@dashoptimization.com
Product Support: support-usa@dashoptimization.com
Tel: +1 (201) 567 9445
Fax: +1 (201) 567 9443
Dash Optimization Inc.
560 Sylvan Avenue
Englewood Cliffs
NJ 07632
USA

## Japan

Dash Optimization Japan
Information and Sales:
info@jp.dashoptimization.com
Licensing:
license@jp.dashoptimization.com
Product Support:
support@jp.dashoptimization.com
Tel: +81 43 297 8836
Fax: +81 43 297 8827
WBG Marive-East 21F FASuC B2124
2-6 Nakase Mihama-ku
261-7121 Chiba
Japan

# Worldwide

Dash Optimization Ltd
Information and Sales: info@dashoptimization.com
Licensing: license@dashoptimization.com
Product Support: support@dashoptimization.com
Tel: +44 1926 315862
Fax: +44 1926 315854
Leam House, 64 Trinity Street
Leamington Spa
Warwickshire CV32 5YN
UK

For the latest news and Xpress-MP software and documentation updates, please visit the Xpress-MP website at http://www.dashoptimization.com or subscribe to our mailing list.

目次

## アブストラクト

このペーパーでは、数理計画法(LPとMIP)によるソリューション・テクニックを、制約計画法（Constraint Programming）と共に使用する例をいくつかのについて説明します。

ここでは、Xpress-Optimizerと、Mosel言語のArtelys-Kalisとを使用します(Moselのモジュールである *mmxprs*、および、*kalis*） 。

最初の例では、LPを解くときに、CPプロパゲーション（CP propagation）を前処理プログラムとして使用し、2番目の例では、CPを、MIPブランチ・アンド・カットアルゴリズムのための、カット生成ヒューリスティクスとして使います。

# 1 イントロダクション

現実の世界が持つ、いろいろな状態や条件を、ますます、厳密に表現することが求められていますが、それらを使用可能な最適化アプリケーションで適切にモデル化して解くことが難しいことがあります。そのような場合、なんらかの分解アプローチ（decomposition approach）をあみ出し、おそらく、問題の一部を異なる方法やツールを利用してモデル化し、解くことが必要になると思います。

モデルを作成し、解くための環境である Xpress-Mosel では、いくつかのソルバー・モジュールを一緒に使って、そのような難問を実行して解くことが出来ます。特に、Xpress-Kalis を、Xpress-Optimizer へのアクセスを可能にする *mmxprs* モジュールと共に使うことで、制約計画法（Constraint Programming (CP)）を線形計画法、混合整数計画法を組み合わせたソリューション・アルゴリズムを実行することができるようになります。

このペーパーでは、問題を解くために、CP を LP/MIP に組み合わせる 2 つのスキームについて議論します。すなわち、

- CP と MIP を使って問題を解くとき、例えば、セクション 2 のプロジェクト計画作成の例に示されているように、まず、LP/MIP 問題の前処理ルーチンとして CP 制約プロパゲーションを使い、次いで、MIP を使う、というように、シークエンシャルに使用する。
- セクション 3 に、パラレル MIP-CP 問題を解く例が説明されていますが、そこでは、MIP ブランチ・アンド・バウンド探索のときに、CP を、マシン割当問題のスケジュール作成での「カット生成ルーチン」として使用する。

このペーパーで説明されている例を実行するには、Xpress-MP の標準インストール(オプティマイザと Mosel)に加え、Xpress-Kalis のインストールが必要です。

また、異なった解法を組み合わせて問題を解くには、関係するソルバーとテクニックに関しての知識も必要です。このペーパーは、読者が、数理計画法、および、制約計画法の両方を理解していること、および、関係する Mosel 言語のソルバーを使った経験を持っていることを前提としています。
Xpress-Optimizer で問題を解く方法の例題については、「Mosel のユーザ・ガイド」、特に、「More about Integer Programming」という章を参照してください。Mosel のモジュール、*mmxprs*、および、*mmjobs* のすべての機能は、「Mosel Language Reference Manual」で説明されています。「Optimizer Reference Manual」には、Dash 社の Optimizer についての詳しい説明があります。また、Dash Optimization 社の Whitepaper「Mosel: multiple models and parallel solving」には、モジュール mmjobs を使用する方法についての、例題を使った説明があります。 Xpress-Kalis についてのドキュメントは、「Xpress-Kalis Reference Manual」にあります。さらに、「Xpress-Kalis User Guide」には、このソフトウェアの使い方についての詳細なイントロダクションがあります。

## 2 前処理プログラムとして CP プロパゲーションを使う

CP モデルの制約式がソルバーに渡されると、制約式は、それに含まれる変数のドメインを減少させるトリガーとなります。簡単なケースでは、イマニュレーション（enumeration） を開始することなく、制約式のプロパゲーションだけで、ソリューションを得ることができる場合もあります。

制約式のプロパゲーションにより得られるドメインの減少は、LP や MIP モデルに渡すことができます。その結果、LP や MIP ソルバーによって使用される前処理アルゴリズムを置き換えたり、補強することができます。

次のセクションの例は、「Applications of optimization with Xpress-MP」という本のセクション 7.1 から取ったものです。

### 2.1 プロジェクトのスケジュール作成の例

町議会は、町に住んでいる人々に提供するサービスを向上させるために、小さいスタジアムを建築したいと計画しています。入札を行った結果、地元の建設会社が落札し、契約を取ることができました。この建設会社は、この建設を、最短時間で完了したいと思っています。主要なタスクは、すべて、下のテーブルにリストアップされています。この表のプロジェクト期間は、週単位で示されています。タスクの中には、別の、あるタスクが完了しないと開始できないものもあります。この表の最後の 2 つのコラムは、下の質問 2 に関係しています。

**質問 1**: 建設を完了できる最早期日は、何時ですか。

**質問 2**:町議会は、建築業者が約束した期日よりも早くプロジェクトを完了させたいと考えています(質問 1 の答)。これを実現するため、町議会は、期日よりも早く完了した場合、1 週間ごとに、30,000 ユーロのボーナスを支払う用意があります。建築業者は、完了までの日数を減らすため、作業員を追加し、建設機械の賃借を増やす必要があります。限界労働者を雇って、合計時により多くの設備をカットに賃借する必要があります。 建築業者は、下表のカラム **Max. reduct.**に示されているように、タスクごとに、最大何週間、短縮できるか、そして、カラム **Add. cost per week (in 1000 )** に示されているように、タスクごと必要な 1 週間あたりの追加費用をまとめました。質問は、建築業者が、彼の利益を最大にしたいと考えて行動する、とすると、プロジェクトは、いつ、完了するか、ということです。

**Table 1**: Data for stadium construction

| Task | Description | Duration | Predecessors | Max. reduct. | Add. cost per week (in 1000 ) |
|---|---|---|---|---|---|
| **1** | Installing the construction site 2 none 0 – | 2 | none | 0 | - |
| **2** | Terracing 16 1 3 30 | 16 | 1 | 3 | 30 |

| | | | | | |
|---|---|---|---|---|---|
| **3** | Constructing the foundations 9 2 1 26 | 9 | 2 | 1 | 26 |
| **4** | Access roads and other networks | 8 | 2 | 2 | 12 |
| **5** | Erecting the basement | 10 | 3 | 2 | 17 |
| **6** | Main floor | 6 | 4, 5 | 1 | 15 |
| **7** | Dividing up the changing rooms | 2 | 4 | 1 | 8 |
| **8** | Electrifying the terraces | 2 | 6 | 0 | - |
| **9** | Constructing the roof | 9 | 4, 6 | 2 | 42 |
| **10** | Lighting of the stadium | 5 | 4 | 1 | 21 |
| **11** | Installing the terraces | 3 | 6 | 1 | 18 |
| **12** | Sealing the roof | 2 | 9 | 0 | - |
| **13** | Finishing the changing rooms | 1 | 7 | 0 | - |
| **14** | Constructing the ticket office | 7 | 2 | 2 | 22 |
| **15** | Secondary access roads | 4 | 4, 14 | 2 | 12 |
| **16** | Means of signaling | 3 | 8, 11, 14 | 1 | 6 |
| **17** | Lawn and sport accessories | 9 | 12 | 3 | 16 |
| **18** | Handing over the building | 1 | 17 | 0 | - |

### 2.2 質問 1 に答えるためのモデルの定式化

CP モデルとしての、この古典的なプロジェクト・スケジューリング問題を表現することはかなり簡単です。まず、プロジェクトの完了に対応するプロジェクト期間ゼロの、架空のタスクを加えます。 その結果、私たちは、*N* がプロジェクトの完了を示す架空のタスクである *TASKS* = {1, . . . , *M*} というタスクの集合を考えます。$DUR_i$ がタスク *i* のプロジェクト期間を示すとしましょう。 タスク間の先行関係は、arcs の集合、すなわち、$ARCS(an \quad arc(i,j) \in ARCS$ で表し、これは、タスク *i* はタスク *j* に先行することを示します。後に続くタスクのないタスクは、すべて、架空のタスクに結ばれます。

ここで、$task_i$ の開始時刻を示す決定変数 $start_i$ を定義します。これらの変数は、間隔 {0, . . . , *HORIZON*} の中の値を取りますが、ここで、*HORIZON* は、すべてのタスクプロジェクト期間の合計によって与えられる総プロジェクト期間の上限です。

こうして、下記のような簡単な CP モデルを得ます。

$$\forall i \in TASKS : start \in \{0,...,HORIZON\}$$

$$\forall (i,j) \in ARCS : start_i + DUR_i \leq start_j$$

### 2.3 質問 1 のインプリメンテーション

このケースの場合、最早完了期日は、明らかに、「架空のタスクN」の最早開始期日です。制約式のプロパゲーションのおかげで、列挙(enumeration)なしで、この値を得ることができます。先行制約式の情報に基づいて、最早完了期日 *bestend* をリトリーブし、最後のタスクに、上限としてこの値を設定します。これにより、クリティカルパス上のタスクの開始期日、完了期日は固定されます。 他のすべてのタスクに関しては、開始期日、完了期日は、実行可能なインターバルに限定されます。

```
model "B-1 Stadium construction (CP)"
 uses "kalis"

 declarations
  N = 19                                    ! Number of tasks in the project
                                            ! (last = fictitious end task)
  TASKS = 1..N
  ARC: array(range,range) of integer        ! Matrix of the adjacency graph
  DUR: array(TASKS) of integer              ! Duration of tasks
  HORIZON : integer                         ! Time horizon

  start: array(TASKS) of cpvar              ! Start dates of tasks
  bestend: integer
 end-declarations

 initializations from 'Data/b1stadium.dat'
  DUR ARC
 end-initializations

 HORIZON:= sum(j in TASKS) DUR(j)

 forall(j in TASKS) do
  0 <= start(j); start(j) <= HORIZON
 end-do

! Task i precedes task j
 forall(i, j in TASKS | exists(ARC(i, j))) do
  Prec(i,j):= start(i) + DUR(i) <= start(j)
  if not cp_post(Prec(i,j)) then
   writeln("Posting precedence ", i, "-", j, " failed")
   exit(1)
  end-if
 end-do

! Since there are no side-constraints, the earliest possible completion
! time is the earliest start of the fictitiuous task N
 bestend:= getlb(start(N))
 start(N) <= bestend
 writeln("Earliest possible completion time: ", bestend)

! For tasks on the critical path the start/completion times have been fixed
! by setting the bound on the last task. For all other tasks the range of
! possible start/completion times gets displayed.
 forall(j in TASKS) writeln(j, ": ", start(j))
```

```
end-model
```

### 2.4 質問 1 の答

上のモデルは、以下の出力を印刷します。 最早完了時間は 64 週間です。 個々のタスクの開始時期、可能な開始時期のインターバルが出力されます。

```
Earliest possible completion time: 64
1: 0
2: 2
3: 18
4: [18..29]
5: 27
6: 37
7: [26..61]
8: [43..59]
9: 43
10: [26..59]
11: [43..58]
12: 52
13: [28..63]
14: [18..53]
15: [26..60]
16: [46..61]
17: 54
18: 63
19: 64
```

### 2.5 質問 2 に答えるためのモデルの定式化

この 2 番目の問題は、「プロジェクトクラッシュでスケジュールを作成する(scheduling with project crashing) 」問題と呼ばれています。プロジェクト期間を短縮するためには、上の最適化ランの結果(*bestend*)を考慮に入れる必要があります。この値は、いまや、すべての開始、完了時間インターバルの上限となりました。

さらに、すべての task $i$ に関して、$MAXW_i$ で、そのタスクが短縮される最大の週の数を示すものとします。

#### 2.5.1 CP モデル

タスク期間は、もはや、固定されておらず、したがって、いまや、決定変数 $duration_i$ によって表され、それらは、範囲 $\{DUR_i - MAXW_i, \ldots, DUR_i\}$ の値を取ります。

これらの新しい変数が追加されたで、CP モデルは、下記のようになります。

$$\forall i \in TASKS : start_i \in \{0,...,bestend\}$$

$$\forall i \in TASKS : duration_i \in \{DUR_i - MAXW_i, ..., DUR_i\}$$

$$\forall (i,j) \in ARCS : start_i + duration_i \leq start_j$$

このモデルは、早くプロジェクトを完了することからの利益の最大化という目的関数を含んでいません。この「目的」は、LP モデルによって対応します(次のセクションを見てください)。CP モデルの制約式プロパゲーションの結果で得られるものは、単に、プロジェクトを完了するのに必要な作業を行うための、切り詰められた開始時刻間隔(reduced start time intervals) です。 タスク期間は、列挙(enumeration)、もしくは、以下に示すように、LP ソリューションアルゴリズムのどちらかで決める必要があります。

### 2.5.2 LP モデル

ここで、task *i* の最も早い開始時期を表す変数 $start_i$を、そして、task *i* で短縮したい週の数に対応する変数 $save_i$ を導入します。与えられている唯一の制約式は、先行制約です。タスク *i* は、すべての先行タスクが完了して、初めて開始できます。このことは、次の制約式で言い換えられます。すなわち、*i* と *j* の間に arc があれば、($start_i$ + $DUR_i$ - $save_i$として計算される) *i* の完了時期は、*i* の開始時期より大きいはずがありません。

$$\forall (i,j) \in ARCS : start_i + DUR_i - save_i \leq start_j$$

変数 $save_i$は、週の数の最大の短縮 $MAXW_i$によって有界です。これらの制約式は、最後の架空のタスク *N*を除き、すべてのタスク *i* で満たされなければなりません。

$$\forall i \in TASKS \setminus \{N\} : save_i \leq MAXW_i$$

最後のタスクに関して、変数 $ave_N$ は、質問 1 に答えて計算された解 bestend よりも、プロジェクトが何週間早く完了するかを示します。プロジェクトの新しい完成時期 *start_{N}*は、 前の完成時期から$save_N$を引いたものに等しくなければなりません。このことから、次の制約式が得られます。

$$start_N = bestend - save_N$$

質問 2 によって定義される目的関数は、建築業者の利益を最大にすることです。 早く終わる 1 週ごとに、建築業者は、*BONUS k* のボーナスを受け取れます。それと引き換えに、*task i* で行われる時間の節約は、$COST_i$ k という費用を発生させます( Table1 の`Add. cost per week'を参照)。こうして、以下の目的関数が得られます。

$$Maximize \;\; BONUS \;\; save_N - \sum_{i \in TASKS \setminus \{N\}} COST_i \;\; save_i$$

数学的モデル全体は、上で説明した制約式と目的関数、および、変数 $start_i$、$save_i$ の非負性の条件から成ります。

$$Maximize\ \ BONUS\ \ \ save_N - \sum_{i \in TASKS \setminus \{N\}} COST_i \ \ \ save_i$$

$$\forall (i,j) \in ARCS : start_i + DUR_i - save_i \le start_j$$

$$start_N = bestend - save_N$$

$$\forall i \in TASKS \setminus \{N\} : save_i \le MAXW_i$$

$$\forall i \in TASKS : start_i \ge 0, \ \ \ save_i \ge 0$$

### 2.6 質問 2 のインプリメンテーション

質問 2 の問題は、以下のアルゴリズムを使って解けます。

- 質問 1 のための CP 問題を解く。そして、ソリューションをリトリーブし、最早完了時期 *bestend* を見つける。
- 時間枠 *bestend* を前提に、質問 2 のための CP 問題を解く。そして、開始時刻インターバルをリトリーブする
- 質問 2 の LP モデルを定義し、そして、CP から得られた start time を LP 変数 $start_i$ の bounds として使い、解く。

インプリメンテーションを行うために、私たちは、上の、質問 1 のソリューションとして示した CP モデルを基礎としたいと思います。 しかし、Kalis solver に示される制約式を変更する方法がないので、一つのファイルで行うことはできません。その代わりとして、インプリメンテーションを 2 つのモデルファイルに分割します。その一つは、アルゴリズムの定義と LP 問題のファイル、そして、もう一つは、CP 問題の定義、および、CP 問題を解くためのファイルです。モデルパラメタ MODE の値によって、この CP モデルは、質問 1、もしくは、質問 2 のモデルを定義します。モデルにインフィージビリティが検出されると、エラーメッセージをプリントする代わりに、CP model は、不成功メッセージ(failure message)を master problem に送ります。「不-実行可能性」が制約式を掲示している間、検出されるならエラーメッセージを印刷することの代わりに、CP モデルは、不成功メッセージをマスター問題に送ります。

```
model "B-1 Stadium construction (CP submodel)"
 uses "kalis", "mmjobs"

 parameters
  MODE = 1                                  ! Model version: 1 - fixed durations
```

```
                                          !                2 - variable dur.
 HORIZON = 100                            ! Time horizon
end-parameters

declarations
 N = 19                                   ! Number of tasks in the project
                                          ! (last = fictitious end task)
 TASKS = 1..N
 ARC: array(range,range) of integer       ! Matrix of the adjacency graph
 DUR: array(TASKS) of integer             ! Duration of tasks
 MAXW: array(TASKS) of integer            ! Max. reduction of tasks (in weeks)

 start: array(TASKS) of cpvar             ! Start dates of tasks
 duration: array(TASKS) of cpvar          ! Durations of tasks

 lbstart,ubstart: array(TASKS) of integer ! Bounds on start dates of tasks
 EVENT_FAILED=2                           ! Event code sent by submodel
end-declarations

initializations from 'Data/b1stadium.dat'
 DUR ARC
end-initializations

forall(j in TASKS) setdomain(start(j), 0, HORIZON)

if MODE = 1 then                          ! **** Fixed durations
 ! Precedence relations between tasks
 forall(i, j in TASKS | exists(ARC(i, j))) do
  Prec(i,j):= start(i) + DUR(i) <= start(j)
  if not cp_post(Prec(i,j)) then
   send(EVENT_FAILED,0)
  end-if
 end-do

 ! Earliest poss. completion time = earliest start of the fictitiuous task N
 start(N) <= getlb(start(N))
else                                      ! **** Durations are variables
 initializations from 'Data/b1stadium.dat'
  MAXW
 end-initializations

 forall(j in TASKS) setdomain(duration(j), DUR(j)-MAXW(j), DUR(j))

 ! Precedence relations between tasks
 forall(i, j in TASKS | exists(ARC(i, j))) do
  Prec(i,j):= start(i) + duration(i) <= start(j)
  if not cp_post(Prec(i,j)) then
   send(EVENT_FAILED,0)
  end-if
 end-do
end-if

! Pass solution data to the master model
forall(i in TASKS) do
 lbstart(i):= getlb(start(i)); ubstart(i):= getub(start(i))
end-do
```

```
 initializations to "raw:"
  lbstart as "shmem:lbstart" ubstart as "shmem:ubstart"
 end-initializations

end-model
```

CP submodel をコンパイルした後に、master model は、まず、質問 1 のためのバージョンをランし、start time intervals をリトリーブします。そして、master model は、今回は、質問 2 向けの形式(in the form for question 2) 、および、時間枠 bestend を与えて、CP submodel を実行します。2 番目の CP ランのソリューションから得られる start time intervals は、それに続く LP 問題の定義で使われます。

```
model "B-1 Stadium construction (CP + LP) master model"
 uses "mmxprs", "mmjobs"

 forward procedure print_CP_solution(version: integer)

 declarations
  N = 19                                      ! Number of tasks in the project
                                              ! (last = fictitious end task)
  TASKS = 1..N
  ARC: array(range,range) of integer          ! Matrix of the adjacency graph
  DUR: array(TASKS) of integer                ! Duration of tasks
  BONUS: integer                              ! Bonus per week finished earlier
  MAXW: array(TASKS) of integer               ! Max. reduction of tasks (in weeks)
  COST: array(TASKS) of real                  ! Cost of reducing tasks by a week
  lbstart,ubstart: array(TASKS) of integer    ! Bounds on start dates of tasks
  HORIZON: integer                            ! Time horizon
  bestend: integer                            ! CP solution value

  CPmodel: Model                              ! Reference to the CP model
  msg: Event                                  ! Termination message sent by submodel
 end-declarations

 initializations from 'Data/b1stadium.dat'
  DUR ARC MAXW BONUS COST
 end-initializations

 HORIZON:= sum(o in TASKS) DUR(o)

! **** First CP model ****

 res:= compile("b1stadium_sub.mos")           ! Compile the CP model
 load(CPmodel, "b1stadium_sub.bim")           ! Load the CP model
 run(CPmodel, "MODE=1,HORIZON=" + HORIZON)    ! Solve first version of CP model
 wait                                         ! Wait until subproblem finishes
 msg:= getnextevent                           ! Get the termination event message
 if getclass(msg)<>EVENT_END then             ! Check message type
  writeln("Submodel 1 is infeasible")
  exit(1)
 end-if
```

```
 initializations from "raw:"
  lbstart as "shmem:lbstart" ubstart as "shmem:ubstart"
 end-initializations

 bestend:= lbstart(N)
 print_CP_solution(1)

! **** Second CP model ****

 run(CPmodel, "MODE=2,HORIZON=" + bestend)  ! Solve second version of CP model
 wait                                       ! Wait until subproblem finishes
 msg:= getnextevent                         ! Get the termination event message
 if getclass(msg)<>EVENT_END then           ! Check message type
  writeln("Submodel 2 is infeasible")
  exit(2)
 end-if

 ! Retrieve solution from memory
 initializations from "raw:"
  lbstart as "shmem:lbstart" ubstart as "shmem:ubstart"
 end-initializations

 print_CP_solution(2)

! **** LP model for second problem ****
 declarations
  start: array(TASKS) of mpvar              ! Start times of tasks
  save: array(TASKS) of mpvar               ! Number of weeks finished early
 end-declarations

! Objective function: total profit
 Profit:= BONUS*save(N) - sum(i in 1..N-1) COST(i)*save(i)

! Precedence relations between tasks
 forall(i,j in TASKS | exists(ARC(i,j)))
  Precm(i,j):= start(i) + DUR(i) - save(i) <= start(j)

! Total duration
 start(N) + save(N) = bestend

! Limit on number of weeks that may be saved
 forall(i in 1..N-1) save(i) <= MAXW(i)

! Bounds on start times deduced by constraint propagation
 forall(i in 1..N-1) do
  lbstart(i) <= start(i); start(i) <= ubstart(i)
 end-do

! Solve the second problem: maximize the total profit
 setparam("XPRS_VERBOSE", true)
 setparam("XPRS_PRESOLVE", 0)   ! We use constraint propagation as preprocessor
 maximize(Profit)

! Solution printing
 writeln("Total profit: ", getsol(Profit))
 writeln("Total duration: ", getsol(start(N)), " weeks")
```

```
 forall(i in 1..N-1)
  write(strfmt(i,2), ": ", strfmt(getsol(start(i)),-3),
   if(i mod 6 = 0,"¥n",""))
 writeln

!*********************************************************************
 procedure print_CP_solution(version: integer)
  writeln("CP solution (version ", version, "):")
  writeln("Earliest possible completion time: ", lbstart(N), " weeks")
  forall(i in 1..N-1)
   write(i, ": ", lbstart(i), if(lbstart(i)<ubstart(i), "-"+ubstart(i), ""),
        if(i mod 6 = 0, "¥n", ", "))
 end-procedure

end-model
```

### 2.7 質問 2 の答

上のモデルは,以下を出力します。XPRS_VERBOSE を true に設定すると、ソフトウェアは、LP ソルバーのログを表示します。ここには、問題のサイズ(制約式の数、変数の数、ノンゼロ係数の数、および、MIP エンティティの数)などの情報と、シンプレクスアルゴリズムに関するログが表示されます。CP から LP への、bound のアップデートなしでモデルをリランすると、わずかですが、Simplex イタレーションの数が増えます。

```
CP solution (version 1):
Earliest possible completion time: 64 weeks
1: 0, 2: 2, 3: 18, 4: 18-29, 5: 27, 6: 37
7: 26-61, 8: 43-59, 9: 43, 10: 26-59, 11: 43-58, 12: 52
13: 28-63, 14: 18-53, 15: 26-60, 16: 46-61, 17: 54, 18: 63
CP solution (version 2):
Earliest possible completion time: 52 weeks
1: 0-12, 2: 2-14, 3: 15-27, 4: 15-37, 5: 23-35, 6: 31-43
7: 21-62, 8: 36-60, 9: 36-48, 10: 21-60, 11: 36-60, 12: 43-55
13: 22-63, 14: 15-57, 15: 21-62, 16: 38-62, 17: 45-57, 18: 51-63

Reading Problem /xprs_6cf5_404d0008
Problem Statistics
          28 (      0 spare) rows
          38 (      0 spare) structural columns
          83 (      0 spare) non-zero elements
Global Statistics
           0 entities        0 sets        0 set members

   Its         Obj Value      S   Ninf   Nneg        Sum Infeas  Time
     0        360.000300      D     17      0      29.000010     0
    17         87.000000      D      0      0        .000000     0
Optimal solution found
Total profit: 87
Total duration: 54 weeks
 1: 0   2: 2   3: 15  4: 15  5: 23  6: 31
 7: 23  8: 36  9: 36 10: 23 11: 36 12: 45
```

13: 25 14: 15 15: 23 16: 39 17: 47 18: 53

## 3 CP と MIP を組み合わせる

現実の適用問題は、いくつかの異なる subproblem を組み合わせたものです。それぞれの subproblem は、そのような問題を得意とするソルバーがあり、したがって、単一のソルバーでは、問題全体を解くのが困難であったり、手に負えないような場合がよくあります。典型的な例は、生産計画作成と生産スケジュール作成のアプリケーションです。 長期(計画作成)の側面は、LP ソルバーにより、通常、容易に扱うことができますが、短期的なスケジュール作成の subproblem を解くには、CP ソルバーのほうが適しています。これらの 2 つの部分を、お互いに、完全に独立して解くと、現実の生産には適さない実行不可能なスケジュール作成の subproblem や計画になってしまう恐れがあります。このジレンマを解消する一つのソリューションは、計画作成の LP 問題と CP によるスケジュール作成問題を、計画数量に見合う実行可能なスケジュールが得られるまで、繰り返し、解くことです。

MIP-CP 問題を解くもう一つの方法は、この二つのテクニックをよりきつく統合する方法で、これは、MIP の Branch-and-Bound サーチのノードで、カットを生成するために、CP subproblem を解くことからなっています。このテクニックは、PSA、および、BASF により、いくつかの大規模な計画作成、スケジュール作成のアプリケーションに使われ、成功しているものです。(例えば、文献[BP03]、[Sad04]の、Mosel を使って実行された hybrid MIP-CP algorithms についての記述を参照してください。)このタイプの組み合わせは、この方法は、アルゴリズムの開発者が、あらゆるノードで MIP サーチと情報のやりとりを行う必要があるという意味で、シークエンシャルに CP と LP/MIP を解く方法よりも、より技術的です。

カット生成アルゴリズムは、Xpress-Optimizer callback を使って実行できます (callbacks with Mosel の定義については、「Mosel Language Reference Manual」を、また、Optimizer callback function については、「Optimizer Reference Manual」を参照してください。)

セクションの例の説明のオリジナルは、[JG99]にあります。このプロトタイプのインプリメンテーションの一つの例は、EU-project LISCOS との関連で、N. Pisaruk により開発されました。

### 3.1 例: マシンの割当てと順序付け

いま、3 台の機械で、12 個の製品を生産する必要があるとします。各マシンは、いずれの製品も生産できますが、処理時間とコストが異なります(Table 2)。さらに、製品のリリース時期と納期が与えられています(Table 3)。ここで、すべての製品のための、生産コストを最小にする生産計画を決定したいと考えています。

**Table 2:** Machine-dependent production costs and durations

| | Production costs | | | Duration | | |
|---|---|---|---|---|---|---|
| Prod./Mach. | 1 | 2 | 3 | 1 | 2 | 3 |
| 1 | 12 | 6 | 7 | 10 | 14 | 13 |
| 2 | 13 | 6 | 10 | 7 | 9 | 8 |
| 3 | 10 | 4 | 6 | 11 | 17 | 15 |
| 4 | 8 | 4 | 5 | 6 | 9 | 12 |
| 5 | 12 | 6 | 7 | 4 | 6 | 10 |
| 6 | 10 | 5 | 6 | 2 | 3 | 4 |
| 7 | 7 | 4 | 5 | 10 | 15 | 16 |
| 8 | 9 | 5 | 5 | 8 | 11 | 12 |
| 9 | 10 | 5 | 7 | 10 | 14 | 13 |
| 10 | 8 | 4 | 5 | 8 | 11 | 14 |
| 11 | 15 | 8 | 9 | 9 | 12 | 16 |
| 12 | 13 | 7 | 7 | 3 | 5 | 6 |

**Table 3:** Release dates and due dates of products

| Product | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Release | 2 | 4 | 5 | 7 | 9 | 0 | 3 | 6 | 11 | 2 | 3 | 4 |
| Due date | 32 | 33 | 36 | 37 | 39 | 34 | 30 | 26 | 36 | 38 | 31 | 22 |

### 3.2 モデルの定式化

ここで、この問題を、機械割当問題と機械の上での処理の順序についての、二つの subproblem で定式化しましょう。前者は、MIP モデル、後者は、CP(単一機械)問題として定式化されます。

#### 3.2.1 MIP モデル

$COSTpm$ により、機械$m$での製品 $p$ の生産コストを、そして、$DURpm$ により、機械$m$での製品 $p$ の処理時間を示すとします。ここで、$p$ は製品の集合（$p \in PRODS$）、$m$は機械の集合（$m \in MACH$）です。

機械割当問題を定式化するために、バイナリー変数 $use_{pm}$ を導入します。ここで、$use_{pm}$ は、機械$m$で製品 $p$ が生産される場合は 1 という値を、機械$m$で製品 $p$ が生産されない場合は 0 という値を取ります。そうすると、目的関数は、下記のように表現できます。

$$\text{minimize} \sum_{p \in PRODS} \sum_{m \in MACH} COST_{pm} \quad use_{pm}$$

割当ての制約式は、各オーダーが必ず一台の機械で処理されるようにするためのもので、下記のように定義されます。

$$\forall p \in PRODS : \sum_{m \in MACH} use_{pm} = 1$$

*MAX_LOAD*: 問題は、既に、完全に記述されていますが、これらの制約式に加えて、LP緩和を強化するためのいくつかの制約式を、追加、定義しましょう。すべての生産は、最も早いリリース期日と最も遅い納入期日の間に行わなければなりません。ここでは、この期間の長さ(最も早いリリース期日と最も遅い納入期日の間)を*MAX_LOAD*で表すとすると、ある機械に割り当てられる製品の処理時間の合計は、*MAX_LOAD*を越えられないということを意味する、下記のような不等式を定式化できます。

$$\forall m \in MACH : \sum_{p \in PRODS} DUR_{pm} \quad use_{pm} \leq MAX\_LOAD$$

### 3.2.2 CP model

ひとたび、機械 *m* に割り当てられる処理の集合、すなわち、$ProdMach_m$ ($ProdMach_m \subseteq PRODS$) がわかると、下記のような、この機械の順序付け問題(sequencing problem) を得ます。

$$\forall p \in ProdMach_m : start_p \in \{REL_p, \ldots, DUE_p - DUR_{pm}\}$$

$$\forall p, q \in ProdMach_m,\ p < q : start_p + DUR_{pm} \leq start_q \vee start_q + DUR_{qm} \leq start_p$$

### 3.3 インプリメンテーション

私たちは、この問題をモデル化し、解くのに、以下のアルゴリズムを使用します。

```
Define the MIP machine assignment problem.
Define the operations of the CP model.
Start the MIP Branch-and-Bound search.
At every node of the MIP search:
  while function generate_cuts returns true
    re-solve the LP-relaxation

Function generate_cuts
  for all machines m call generate_cut_machine(m)
  if at least one cut has been generated
    Return true
  otherwise
    Return false

Function generate_cut_machine(m)
  Collect all operations assigned to machine m
  if more than one operation assigned to m
    Solve the CP sequencing problem for m
    if sequencing succeeds
```

```
    Save the solution
  otherwise
    Add an infeasibility cut for machine m to the MIP
```

このモデルのインプリメンテーションは、二つの Mosel モデルに分けられます。最初のモデル sched_main.mos は、MIP マスター問題とカット生成アルゴリズムの定義を含んでいます。二番目のモデル sched_sub.mos は、CP の単一マシンの順序付けモデルを行います。

マスターモデルの最初の部分は、データ配列をセットアップして、CP submodel をコンパイルして、ロードし、モデル定義と解くためのサブルーチンを呼び出して、終わりに、結果のサマリーを出力します。ここでは、データファイルのファイル名をパラメタと定義し、モデルの実行のときに、モデルのソースを変えることなく、データファイルの名前を変えることができるようにしてあります。同様に、インデックスセットのサイズを含む、すべてのデータは、ファイルから読み込まれます。 初めに、NP と NM の値だけを読みます。続いて、これらの値を利用するセットと配列を宣言(declare)するとき、NP と NM は既知で、したがって、配列は固定配列として作成されます。そうしておかないと、インデックスセットが判らないので、これらの配列は、自動的に、dynamic array として宣言され、basic type の array (real, integer, etc.)以外のすべての配列のエントリーを、明示的に作成しなければなりません。

```
model "Schedule (MIP + CP) master problem"
 uses "mmsystem", "mmxprs", "mmjobs"

 parameters
  DATAFILE = "Data/sched_3_12.dat"
  VERBOSE = 1
 end-parameters

 forward procedure define_MIP_model
 forward procedure setup_cutmanager
 forward public function generate_cuts: boolean
 forward public procedure print_solution

 declarations
  NP: integer                                ! Number of operations (products)
  NM: integer                                ! Number of machines
 end-declarations

 initializations from DATAFILE
  NP NM
 end-initializations

 declarations
  PRODS = 1..NP                              ! Set of products
  MACH = 1..NM                               ! Set of machines

  REL: array(PRODS) of integer               ! Release dates of orders
  DUE: array(PRODS) of integer               ! Due dates of orders
  MAX_LOAD: integer                          ! max_p DUE(p) - min_p REL(p)
  COST: array(PRODS,MACH) of integer         ! Processing cost of products
  DUR: array(PRODS,MACH) of integer          ! Processing times of products
```

```
  starttime: real                          ! Measure program execution time
  ctcut: integer                           ! Counter for cuts
  solstart: array(PRODS) of integer
                                           ! **** MIP model:
  use: array(PRODS,MACH) of mpvar          ! 1 if p uses machine m, otherwise 0
  Cost: linctr                             ! bjective function

  totsolve,totCP: real                     ! Time measurement
  ctrun: integer                           ! Counter of CP runs
  CPmodel: Model                           ! Reference to the CP sequencing model
  ev: Event                                ! Event
  EVENT_SOLVED=2                           ! Event codes sent by submodels
  EVENT_FAILED=3
 end-declarations

 ! Read data from file
 initializations from DATAFILE
  REL DUE COST DUR
 end-initializations

! **** Problem definition ****
 define_MIP_model                          ! Definition of the MIP model
 res:=compile("sched_sub.mos")             ! Compile the CP model
 load(CPmodel, "sched_sub.bim")            ! Load the CP model

! **** Solution algorithm ****
 starttime:= gettime
 setup_cutmanager                          ! Settings for the MIP search

 totsolve:= 0.0
 initializations to "raw:"
  totsolve as "shmem:solvetime"
  REL as "shmem:REL" DUE as "shmem:DUE"
 end-initializations

 minimize(Cost)                            ! Solve the problem

 writeln("Number of cuts generated: ", ctcut)
 writeln("(", gettime-starttime, "sec) Best solution value: ", getobjval)
 initializations from "raw:"
  totsolve as "shmem:solvetime"
 end-initializations
 writeln("Total CP solve time: ", totsolve)
 writeln("Total CP time: ", totCP)
 writeln("CP runs: ", ctrun)
```

MIP モデルは、前のセクションで見た数学的モデルに、厳密に対応しています。

```
 procedure define_MIP_model

 ! Objective: total processing cost
  Cost:= sum(p in PRODS, m in MACH) COST(p,m) * use(p,m)

 ! Each order needs exactly one machine for processing
  forall(p in PRODS) sum(m in MACH) use(p,m) = 1
```

```
! Valid inequalities for strengthening the LP relaxation
 MAX_LOAD:= max(p in PRODS) DUE(p) - min(p in PRODS) REL(p)
 forall(m in MACH) sum(p in PRODS) DUR(p,m) * use(p,m) <= MAX_LOAD

 forall(p in PRODS, m in MACH) use(p,m) is_binary

end-procedure
```

カット生成のための callback function generate_cuts は、MIP のノードごとに、少なくとも一度はコールされます。すべての機械にたいして、それは、割り当てられた処理が予定できるかどうか、または、インフィージビリティカットを加える必要があるかどうかをチェックします。カットが加えられると、LP 緩和を、再度、解くことが必要になり、そして、cut generation function が、カットが、それ以上、全く加えられなくなるまで、コールされます。Xpress-Optimizer のソリューションの値にアクセスする必要があるなら、この例に示されているように、このファンクションの最初、および、最後で、XPRS_solutionfile の値をセットし、リセットすることが重要ですので気をつけてください。

generate_cut_machine ファンクションは、まず、procedure products_on_machine をコールして、所与の機械 $m$ に割り当てられている、すべてのタスクを、set ProdMach に入れます。いまだ、割り当てられていないタスクがあると、帰されてくるセットは空で、もし、そうではなく、セットが一つでも要素を持っていると、それは、sequencing subproblem (function solve_CP_problem)を解こうとします。この問題を解くことができないと、このファンクションは、この機械への、現在の割当てをインフィージブルにしている MIP 問題にカットを追加します。

```
procedure products_on_machine(m: integer, ProdMach: set of integer)

 forall(p in PRODS) do
  val:=getsol(use(p,m))
  if (val > 0 and val < 1) then
  ProdMach:={}
   break
  elif val>0.5 then
   ProdMach+={p}
  end-if
 end-do

end-procedure


!-------------------------------------------------------------------
! Generate a cut for machine m if the sequencing subproblem is infeasible
 function generate_cut_machine(m: integer): boolean
  declarations
   ProdMach: set of integer
  end-declarations

 ! Collect the operations assigned to machine m
  products_on_machine(m, ProdMach)
```

```
 ! Solve the sequencing problem (CP model): if solved, save the solution,
 ! otherwise add an infeasibility cut to the MIP problem
  size:= getsize(ProdMach)
  returned:= false
  if (size>1) then
   if not solve_CP_problem(m, ProdMach, 1) then
    Cut:= sum(p in ProdMach) use(p,m) - (size-1)
    if VERBOSE > 2 then
     writeln(m,": ", ProdMach, " <= ", size-1)
    end-if
    addcut(1, CT_LEQ, Cut)
    returned:= true
   end-if
  end-if

  end-function

 !-------------------------------------------------------------------
 ! Cut generation callback function
  public function generate_cuts: boolean
   returned:=false; ctcutold:=ctcut

   setparam("XPRS_solutionfile", 0)
   forall(m in MACH) do
    if generate_cut_machine(m) then
     returned:=true                          ! Call function again for this node
     ctcut+=1
    end-if
   end-do
   setparam("XPRS_solutionfile", 1)
   if returned and VERBOSE>1 then
    writeln("Node ", getparam("XPRS_NODES"), ": ", ctcut-ctcutold,
            " cut(s) added")
   end-if

  end-function
```

function solve_CP_problem は、CP モデルを解き始め、必要なデータを共有メモリに書き出し、ファイル sched_sub.mos の中にある submodel の実行を開始します。

```
 function solve_CP_problem(m: integer, ProdMach: set of integer,
                           mode: integer): boolean
  declarations
   DURm: array(range) of integer
   sol: array(range) of integer
   solvetime: real
  end-declarations

 ! Data for CP model
  forall(p in ProdMach) DURm(p):= DUR(p,m)
  initializations to "raw:"
   ProdMach as "shmem:ProdMach"
   DURm as "shmem:DURm"
  end-initializations
```

```
! Solve the problem and retrieve the solution if it is feasible
 startsolve:= gettime
 returned:= false
 if(getstatus(CPmodel)=RT_RUNNING) then
   fflush
   writeln("CPmodel is running")
   fflush
   exit(1)
 end-if

 ctrun+=1
 run(CPmodel, "NP=" + NP + ",VERBOSE=" + VERBOSE + ",MODE=" + mode)
 wait                                          ! Wait for a message from the submodel
 ev:= getnextevent                             ! Retrieve the event
 if getclass(ev)=EVENT_SOLVED then
  returned:= true
  if mode = 2 then
   initializations from "raw:"
    sol as "shmem:solstart"
   end-initializations
   forall(p in ProdMach) solstart(p):=sol(p)
  end-if
 elif getclass(ev)<>EVENT_FAILED then
  writeln("Problem with Kalis")
  exit(2)
 end-if
 wait
 dropnextevent                                 ! Ignore "submodel finished" event
 totCP+= (gettime-startsolve)
end-function
```

ここで、cut manager のセッティング、および、integer solution callback の定義で MIP モデルを終えます。Mosel の comparison tolerance は、Xpress-Optimizer によって適用されるトレランスよりも、わずかに大きい値に設定されます。LP の presolve を off にすることが重要です。なぜなら、アルゴリズムの実行の間、マトリクスに干渉するからです。代替的に、presolve を、non-destructive algorithms のみを使うように微調整することも可能です。また、マトリクスの中に、カットとカット係数のための十分大きいスペースを用意しておかなければなりません。また、Xpress-Optimizer による出力のプリンティングを可能にし、MIP ログの頻度を選びます(モデルパラメタ VERBOSE) 。

```
procedure setup_cutmanager
 setparam("XPRS_CUTSTRATEGY", 0)               ! Disable automatic cuts
 feastol:= getparam("XPRS_FEASTOL")            ! Get Optimizer zero tolerance
 setparam("zerotol", feastol * 10)             ! Set comparison tolerance of Mosel
 setparam("XPRS_PRESOLVE", 0)                  ! Disable presolve
 setparam("XPRS_MIPPRESOLVE", 0)               ! Disable MIP presolve
 command("KEEPARTIFICIALS=0")                   No global red. cost fixing
 setparam("XPRS_SBBEST", 0)                    ! Turn strong branching off
```

```
 setparam("XPRS_HEURSTRATEGY", 0)            ! Disable MIP heuristics
 setparam("XPRS_EXTRAROWS", 10000)           ! Reserve space for cuts
 setparam("XPRS_EXTRAELEMS", NP*30000)       ! ... and cut coefficients
 setcallback(XPRS_CB_CM, "generate_cuts")    ! Define the cut manager callback
 setcallback(XPRS_CB_UIS, "print_solution")  ! Define the integer solution cb.
 setparam("XPRS_COLORDER", 2)
 case VERBOSE of
 1: do
     setparam("XPRS_VERBOSE", true)
     setparam("XPRS_MIPLOG", -200)
    end-do
 2: do
     setparam("XPRS_VERBOSE", true)
     setparam("XPRS_MIPLOG", -100)
    end-do
 3: do                                       ! Detailed MIP output
     setparam("XPRS_VERBOSE", true)
     setparam("XPRS_MIPLOG", 3)
    end-do
 end-case

end-procedure
```

整数ソリューションの callback の定義は、部分的に function generate_cut_machine と似ています。詳細なソリューション出力を得るためには、すべての CP subproblem を解きなおす必要がありますが、今回は、run MODE two で行います。これは、CP モデルが解の情報を共有メモリに書くことを意味します。

```
public procedure print_solution
 declarations
  ProdMach: set of integer
 end-declarations

 writeln("(",gettime-starttime, "sec) Solution ",
         getparam("XPRS_MIPSOLS"), ": Cost: ", getsol(Cost))

 if VERBOSE > 1 then
  forall(p in PRODS) do
   forall(m in MACH) write(getsol(use(p,m))," ")
   writeln
  end-do
 end-if

 if VERBOSE > 0 then
  forall(m in MACH) do
   ProdMach:= {}

 ! Collect the operations assigned to machine m
   products_on_machine(m, ProdMach)

   Size:= getsize(ProdMach)
   if Size > 1 then
  ! (Re)solve the CP sequencing problem
```

```
    if not solve_CP_problem(m, ProdMach, 2) then
     writeln("Something wrong here: ", m, ProdMach)
    end-if
   elif Size=1 then
    elem:=min(p in ProdMach) p
    solstart(elem):=REL(elem)
   end-if
  end-do

! Print out the result
  forall(p in PRODS) do
   msol:=sum(m in MACH) m*getsol(use(p,m))
   writeln(p, " -> ", msol,": ", strfmt(solstart(p),2), " - ",
           strfmt(DUR(p,round(msol))+solstart(p),2), "  [",
           REL(p), ", ", DUE(p), "]")
  end-do
  writeln("Time: ", gettime - starttime, "sec")
  writeln
  fflush
 end-if
end-procedure
```

以下のコードリストは、CP submodel の全部をリストしています。実行のときには、いつでも、1 台のマシンに割り当てられたタスクのセット、および、対応するプロジェクト期間は共有メモリから読み込まれます。問題の定義の間にインフィージビリティが検出されると、「pairs of tasks」の間の「disjunctions」が明確に掲示され、制約式の追加を止めることがでます。サーチは、最初の実現可能解で止まります。解が見つかると、モデル・パラメタ MODE が値 2 であるなら、それは master model に送られます。あらゆる場合で、CP サーチの終了の後に、submodel は、解のステータスを、master model に返えします。

```
model "Schedule (MIP + CP) CP subproblem"
 uses "kalis", "mmjobs" , "mmsystem"

 parameters
  VERBOSE = 1
  NP = 12                                  ! Number of products
  MODE = 1                                 ! 1 - decide feasibility
                                           ! 2 - return complete solution
 end-parameters

 startsolve:= gettime

 declarations
  PRODS = 1..NP                            ! Set of products
  ProdMach: set of integer
 end-declarations

 initializations from "raw:"
  ProdMach as "shmem:ProdMach"
 end-initializations
```

```
 finalize(ProdMach)

 declarations
  REL: array(PRODS) of integer               ! Release dates of orders
  DUE: array(PRODS) of integer               ! Due dates of orders
  DURm: array(ProdMach) of integer           ! Processing times on machine m
  solstart: array(ProdMach) of integer       ! Solution values for start times

  start: array(ProdMach) of cpvar            ! Start times of tasks
  Disj: array(range) of cpctr                ! Disjunctive constraints
  Strategy: array(range) of cpbranching      ! Enumeration strategy
  EVENT_SOLVED=2                             ! Event codes sent by submodels
  EVENT_FAILED=3
  solvetime: real
 end-declarations

 initializations from "raw:"
  DURm as "shmem:DURm" REL as "shmem:REL" DUE as "shmem:DUE"
 end-initializations

! Bounds on start times
 forall(p in ProdMach) setdomain(start(p), REL(p), DUE(p)-DURm(p))

! Disjunctive constraint
 ct:= 1
 forall(p,q in ProdMach| p<q) do
  Disj(ct):= start(p) + DURm(p) <= start(q) or start(q) + DURm(q) <= start(p)
  ct+= 1
 end-do

! Post disjunctions to the solver
 nDisj:= ct; j:=1; res:= true
 while (res and j<nDisj) do
  res:= cp_post(Disj(j))
  j+=1
 end-do

! Solve the problem
 if res then
  Strategy(1):= settle_disjunction(Disj)
  Strategy(2):= assign_and_forbid(KALIS_SMALLEST_DOMAIN, KALIS_MIN_TO_MAX,
                                  start)
  cp_set_branching(Strategy)
  res:= cp_find_next_sol
 end-if

! Pass solution to master problem
 if res then
  forall(p in ProdMach) solstart(p):= getsol(start(p))
  if MODE=2 then
   initializations to "raw:"
    solstart as "shmem:solstart"
   end-initializations
  end-if
  send(EVENT_SOLVED,0)
```

```
 else
  send(EVENT_FAILED,0)
 end-if

! Update total running time measurement
 initializations from "raw:"
  solvetime as "shmem:solvetime"
 end-initializations
 solvetime+= gettime-startsolve
 initializations to "raw:"
  solvetime as "shmem:solvetime"
 end-initializations

end-model
```

### 3.4 結果

下記は、sched_3_12 の最も良い解です。

```
Cost: 92
1 -> 3:  2 - 15  [2, 32]
2 -> 3: 15 - 23  [4, 33]
3 -> 2: 15 - 32  [5, 36]
4 -> 1: 24 - 30  [7, 37]
5 -> 2: 32 - 38  [9, 39]
6 -> 2:  0 -  3  [0, 34]
7 -> 1:  3 - 13  [3, 30]
8 -> 1: 16 - 24  [6, 26]
9 -> 3: 23 - 36  [11, 36]
10 -> 1: 30 - 38  [2, 38]
11 -> 2:  3 - 15  [3, 31]
12 -> 1: 13 - 16  [4, 22]
```

2691 回の CP モデルのランで、合計 1604 のカットが MIP 問題に加えられ、Branch-and-Bound サーチは、12295 のノードを探りました。Optimality は、Pentium IV PC 上で、数秒以内に証明されました。

この問題のインプリメンテーションは、Xpress-Optimizer か Xpress-Kalis で、 完全に行えます。しかし、この 3 台の機械、12 の仕事を解くことは、既に、Xpress-Optimizer、Xpress-Kalis にとっても、それだけでは、解くのが非常に難しい問題です。CP では、optimality を証明するのが難しく、また、MIP では、disjunction の定式化は、多数(およそ、number_of_machines * number_of_products[2)] のバイナリー変数の定義が必要なので、このまま、この問題を扱うのは、非現実的です。

### 3.5 CP subproblem をパラレルに解く

そこで、あらゆる MIP ノードで、CP single-machine subproblem を順番に解くことの代わりに、Mosel モデルを変更し、パラレルに subproblem を解くようにします。特に、multiprocessor machine 環境で仕事をしていると、これにより、カット生成のプロセスをスピードアップし、したがって、ランタイムの合計を短縮できます。こうして、セクション 3.3 のアルゴリズムを、以下のように変更します。

```
Define the MIP machine assignment problem.
Define the operations of the CP model.
Start the MIP Branch-and-Bound search.
At every node of the MIP search:
  while function generate_cuts returns true
    re-solve the LP-relaxation

Function generate_cuts
  Collect all machines that are fully assigned into set ToSolve
  for all machines m  ¥in  ToSolve call start_CP_model(m)
  Wait for the solution status messages from all submodels
   if submodel m is infeasible
     Add an infeasibility cut for machine m to the MIP
  if at least one cut has been generated
    Return true
  otherwise
    Return false

Procedure start_CP_model(m)
  Collect all operations assigned to machine m
  Write data for this machine to memory
  Start the submodel execution
```

function generate_cuts の変更バージョンは、以下のようになるでしょう。なお、full example code をご覧になりたい読者の皆さんは、Xpress-Kalis と共に配布(files sched_mainp.mos and sched_subp.mos)されている User Guide examples を参照してください。

```
 procedure products_on_machine(m: integer)

  NumOp(m):=0
  forall(p in PRODS) do
   val:=getsol(use(p,m))
   if (! not isintegral(use(p,m)) !) (val > 0 and val < 1) then
    NumOp(m):=0
    break
   elif val>0.5 then
    NumOp(m)+=1
    OpMach(m,NumOp(m)):= p
   end-if
  end-do

 end-procedure

!-------------------------------------------------------------------
! Add an infeasibility cut for machine m to the MIP problem
 procedure add_cut_machine(m: integer)

  Cut:= sum(p in 1..NumOp(m)) use(OpMach(m,p),m) - (NumOp(m)-1)
  if VERBOSE > 1 then
   write(m,": ")
   forall(p in 1..NumOp(m)) write(OpMach(m,p), " ")
```

```
 writeln(" <= ", NumOp(m)-1)
end-if
addcut(1, CT_LEQ, Cut)
```

CP submodels のインプリメンテーションは、その多くは変わりませんが、例外は、共有メモリを経由してデータをパスするのに使われるレイベルです。すなわち、machine index をあらゆるデータ項目に追加し、パラレルにランしている、異なる subproblem によって使用されるデータを見分けることができるようにします。

データセット sched_3_12.dat に関して見ると、パラレル・インプリメンテーションを使う dual processor machine 上で、総実行時間の減少は、ほんの数パーセントに過ぎないことを観測しました。これは、この問題では、多くのノードで、たった一つの CP subproblem しか解かれないことによるもので、もし、いくつかの subproblem を解くとするなら、それらの実行は、非常に短縮されたものになるでしょう。機械の数が三台よりももっと多く、製品数も多ければ、並行処理(parallelization) は、もっと、大きな効果を示すでしょう。

## サマリー

この whitepaper の例で、CP と LP/MIP を組み合わせて、モデルを作成し、問題を解くための二つの scheme を示しました。一番目の例は、CP 問題と LP 問題を緩く結合して、順繰りに解く結合アルゴリズムですが、二番目の例は、MIP branch-and-cut search のカット生成アルゴリズムとして CP を使用する、なりきつい統合の例です。他の多くの scheme が可能です。例えば、[Tim02]に示されているような、一連の MIP と CP subproblem を、繰り返し解くというような方法です。hybridization scheme は、個々の適用問題が持つ、特定の構造と、その典型的なデータ例(typical data instances) によって、選ばなければなりません。

異なる解法テクニックの組み合わせは、多くのアプリケーションで成功しましたが、著者は、興味を持つ読者の皆さんに、一つのテクニックで取り扱える問題に、この考え方を適用することは、多くの場合、時間のムダであるということをお伝えして置きたいと思います。ハイブリッド・ソリューション・アルゴリズムは、ケース・バイ・ケースで、開発され、実行され、テストされる必要があります。これは、開発努力でのかなり大きな投資が必要であること、そして、関係する解法とソルバーについての深い理解が必要であることを意味します。

## Bibliography

**[JG99]** Jain, V. and Grossmann, I.E.: "Algorithms for hybrid MILP/CLP models for a class of optimization problems". INFORMS J. Computing, 13(4), pp. 258 - 276, 2001.

**[BP03]** Bockmayr, A. and Pisaruk, N.: "Detecting Infeasibility and Generating Cuts

for MIP Using CP". In: Proceedings of CP-AI-OR 2003. Montreal, pp. 24 - 34, 2003.

[Sad04] Sadykov, R.: "A Hybrid Branch-and-Cut Algorithm for the One-Machine Scheduling Problem". Ed. R'egin, J.C. and Rueher, M., In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2004. Springer, Berlin, pp. 409 - 414, 2004.

[Tim02] Timpe, C.: "Solving planning and scheduling problems with combined integer and constraint programming". ORSpectrum, 24(4), pp. 431 - 448, 2002.