スケジューリング最適化ソルバー OptSeq II (Ver. 2.3)

導入ガイド

LOG OPT Co., Ltd.

ご注意

目 次

1　はじめに

本ドキュメントでは，スケジューリング最適化システム OptSeq II で対象とする「モデル」について解説します．通常の処理的なソフトウェアで用いられているディスパッチング・ルールや山崩し法とよばれる簡便法は，OptSeq II のソルバーとくらべて極端に悪いスケジュールを生成します．また，OptSeq II は，メタヒューリスティクスとよばれる最新技術を用いているので，求解時間に応じて，さらに良い解を探索することができます．

OptSeq II のトライアル・バージョンは，LOGOPT 社のホームページ

`http://www.logopt.com/OptSeq/OptSeq.htm`

から無料でダウンロードして試用することができます．このトライアル・バージョンは，作業数 15 までの問題を解くことができます．また，本ドキュメントで使用されたプログラムも，同じ場所からダウンロードできます．

以下では，モデルについて例題を交えて丁寧に解説すると同時に，実際問題をモデルに帰着させるための様々なテクニックを紹介します．

2　スケジューリングモデルとは

一般に，スケジューリングモデルは，以下の基本的な要素から構成されます．

作業： 作業とは，成すべき仕事のことです．しばしば，作業はジョブ，タスク，オペレーション，**活動**（activity）などとよばれます．作業には，その仕事を行うときにかかる時間（作業量）や，その仕事を終わらさなければならない期限（納期）などの情報が付加されます．

先行（時間）制約： ある作業がある作業の前に処理されなければならない条件を**先行制約**（precedence constraint）もしくは**時間制約**（temporal constraint）とよびます．多くのスケジューリングモデルでは，作業間に先行制約が付加されます．たとえば,「のどが渇いたのでジュースを買って飲む」ためのスケジューリングモデルでは,「ジュースを買う」,「蓋をあける」,「ジュースを飲む」の 3 つの作業がありますが，これらの作業の間には，ジュースを買う前に蓋を開けることはできないこと，蓋を開ける前にジュースを飲むことができないこと，を表す 2 つの先行制約を満たす必要があります.（ここで，ジュースを飲む前にジュースを買わなければならないことは，ここで定義した 2 つの先行制約から導かれるので，考慮しなくても大丈夫です.）

資源： 作業を行うときに必要なもので，その量に限りがあるものを**資源**（resource）とよびます．資源はしばしばリソース，機械などとよばれることがあります．資源は，作業を行う人の稼働できる時間や，機械そのものや，製品を作るための材料などを表す抽象的な概念です．現場に応じて，何を資源としてモデルに組み込むかは，モデルを作成する人のセンスに依存します．実際には，その量に限りがあっても，その製造現場では制約にならないほど十分にあるものは,（少なくとも OptSeq II で最適化を行う際には）資源としてモデルに入れる必要はありません．

他にもスケジューリングモデルには，様々な要素が入ってきますが，以下で例題と練習問題を通して順次解説していきます．

以下の構成は次にようになっています．

§ 3 では，プロジェクトスケジューリングの古典モデルである PERT（program evaluation and review technique）を例として，先行制約の使い方を学びます．

§ 4 では，資源制約付きの PERT の例題を通して，資源のモデル化方法を説明します．また，サブルーチンを使って，記述を簡潔にする方法について学びます．

§ 5 では，F1 のピットイン時間の短縮の例題を通して，並列ショップスケジューリングとよばれる問題のモデル化を解説します．

§ 6 では，前節と同じ F1 のピットイン時間の短縮の例題を通して，作業のモードの概念を説明します．

§ 7 では，一般の資源制約付きスケジューリング問題の OptSeq II を用いたモデル化について解説します．

§ 8 では，納期遅れの最小化について解説します．

§ 9 では，モードと再生不能資源を用いて，クリティカルパス法とよばれる PERT の拡張を例として，帰着のテクニックをご紹介します．特に，再生可能資源（通常の機械や人のような資源）と再生不能資源（お金や原料のような資源）の違いを解説し，再生不能資源を表現する方法を説明します．

§ 10 では，先行制約の一般化と，それを用いた種々の作業間のタイミングの設定法について解説します．

§ 11 では，作業を途中で中断することができる場合の記述法について説明します．

§ 12 では，作業の並列処理を簡単に記述する方法について解説します．

3 PERT–航空機を早く離陸させよう!–

ここで学ぶこと

- 作業と先行制約の入力の仕方
- 最大完了時刻（すべての作業が終了する時刻）を表現する方法
- ソルバーの起動方法とソルバーオプション
- 結果の読み方とファイルに出力する方法

最初の例題として，**PERT**（Program Evaluation and Review Technique）および **CPM**（Critical Path Method ; クリティカルパス法）とよばれる，スケジューリング理論の始祖とも言える古典的なモデルを考えてみましょう．ちなみに，PERT は，第 2 次世界大戦中における米国海軍のポラリス潜水艦に搭載するミサイルの設計・開発時間の短縮に貢献したことで有名になって，その後オペレーションズ・リサーチの技法の代表格として知られています．現在では，PERT は，多くのプロジェクト・スケジューリングのためのソフトウェアに導入されており，一般に普及しています．クリティカルパス法については，§ 9 で詳述します．

あなたは航空機会社のコンサルタントだ．あなたの仕事は，着陸した航空機をなるべく早く離陸させるためのスケジュールをたてることだ．航空機は，再び離陸する前に幾つかの作業をこなさなければならない．まず，乗客と荷物を降ろし，次に機内の掃除をし，最後に新しい乗客を搭乗させ，新しい荷物を積み込む．当然のことであるが，乗客を降ろす前に掃除はできず，掃除をした後でないと新しい乗客を入れることはできず，荷物をすべて降ろし終わった後でないと，新しい荷物は積み込むことができない．また，この航空機会社では，乗客用のゲートの都合で，荷物を降ろし終わった後でないと新しい乗客を搭乗させることができないのだ．作業時間は，乗客降ろし 13 分，荷物降ろし 25 分，機内清掃 15 分，新しい乗客の搭乗 27 分，新しい荷物積み込み 22 分とする．さて，最短で何分で離陸できるだろうか?

この問題に含まれる作業とそれを行うのに必要な時間（作業時間）は，以下のようにまとめられます．

作業 1：乗客降ろし （13 分）

作業 2：荷物降ろし （25 分）

作業 3：機内清掃 （15 分）

作業 4：新しい乗客の搭乗 （27 分）

作業 5：新しい荷物積み込み （22 分）

作業間の先行制約は，図 1 のようになっています．この図には，すべての作業が終了したあとに航空機が飛び立つことを表す作業（離陸）が追加されています．このように，もとの問題には明示的には含まれていないが，モデル内で追加する仮想の対象を「ダミー」とよびます．この離陸を表すダミー作業をなるべく早く終了させることが，問題の目的となります．

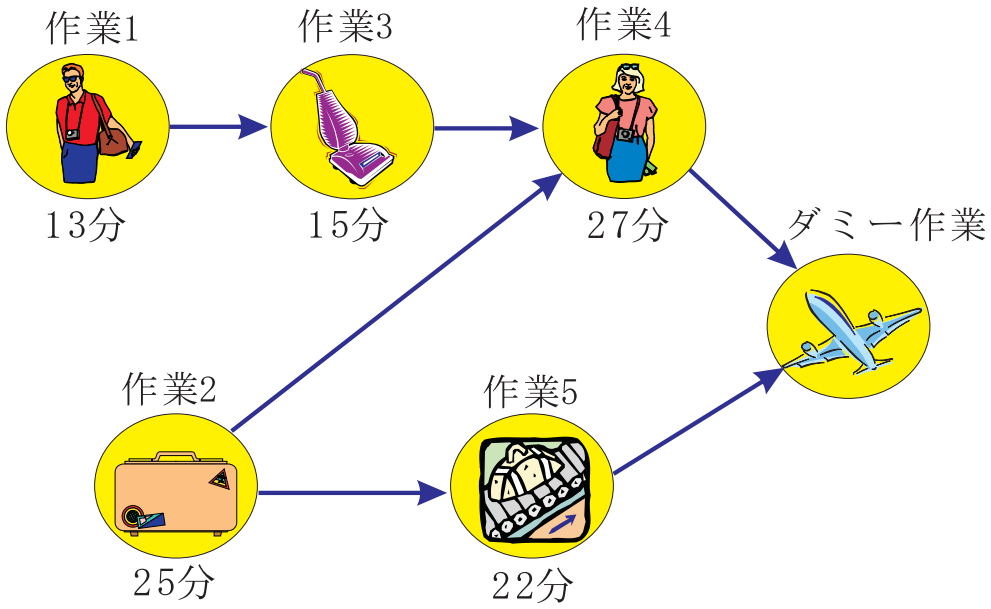

図 1: 作業時間と作業間の先行制約

この例題を OptSeq II で解いてみましょう.

作業を入力します．作業を定義するためのキーワードは，activity です．作業に関するデータとして作業時間がありますが，OptSeq II では作業に付随するモード（mode）を定義し，モードに対して使用する資源と作業時間を入力します.（ここでは資源は使用しません.）

作業の定義は

```
activity  作業名
   モード名    モード名       ...
```

と行います（注意：モード名には “dummy” 以外の文字列を入力してください）.

モードが 1 つのときには，モード名を省略して，mode の後ろに作業時間を表すキーワード duration をつけて作業時間を設定することができます.

```
activity  作業名
   mode duration 作業時間
```

また，作業名の後ろに，キーワード duedate を付加することによって，作業の納期を設定できます.

```
activity  作業名 duedate 納期
```

OptSeq II では，作業の完了時刻が，納期を超過した量（納期遅れ）を最小にするようなスケジュールを探索します.

例題では，5 つの作業を以下のように定義します.

```
activity activity[1]
        # 処理モードが 1 つだけのときは，activity 記述内に mode を記述可能
        mode  duration 13


activity activity[2]
        mode  duration 25


activity activity[3]
        mode  duration 15


activity activity[4]
        mode  duration 27


activity activity[5]
        mode  duration 22
```

ここで，# の右側はコメント文なので，無視されます．

続いて，作業間の先行制約（乗客を降ろす前に掃除はできず，掃除をした後でないと新しい乗客を入れることはできず，荷物をすべて降ろし終わった後でないと新しい荷物は積み込むことができないことを表す制約）を入力します．

先行制約の定義は

```
temporal  先行作業  後続作業
```

と行います．これによって，先行作業が完了した後に，後続作業を行うという制約が付加されます．

例題の先行制約は，以下のように定義します．

```
temporal  activity[1] activity[3]
temporal  activity[2] activity[4]
temporal  activity[2] activity[5]
temporal  activity[3] activity[4]
```

先行制約には，**段取り時間**（setup time；作業が終わってから次の作業を開始する前にかかる準備時間）を付加することもできます．たとえば，作業 1 の後，作業 3 を開始する前に，5 分の段取り時間が必要な場合には，キーワード delay を使って，

```
temporal  activity[1] activity[3] delay 5
```

と書きます．

段取り時間は，負の値を設定することも可能です．たとえば，段取り時間に -5 分を入れた場合には，作業 1 が終了する 5 分前以降に作業 2 を開始しなければならないことを表します．

以上で，作業（および付随するモード），ならびに先行制約が入力されました．最後に，目的である最後の作業が完了する時間を最小化することを入力しましょう．

OptSeq II では，すべての作業に後続するダミーの作業 sink が定義されています．(ちなみに，すべての作業に先行するダミーの作業 source もあります．）したがって，最後に終了する作業の完了時刻を最小にするためには，ダミーの作業 sink の開始時刻をなるべく小さくするように設定します．入力は，

```
activity sink duedate 0
```

とします．

以上でデータ入力が完了しましたので，いよいよ求解してみましょう．

Windows の場合には，コマンドプロンプトを起動して，保存したファイルと実行ファイル `optseq.exe` が保存されているフォルダに移動してください．問題例のデータは標準入力から読み込まれます．データを記述した入力ファイル名を *inputfile* として，

`optseq <` *inputfile*

で実行されます.

OptSeq II の実行の際には，様々なオプションを設定することができます.

```
optseq -help
```

とすると，

```
Usage: optseq [-options...]
  -backtrack #  set max number of backtrack (default: 100)
  -data         print input data and terminate immediately
  -initial f    set initial solution file (default: not specified)
  -iteration #  set iteration limit (default: 1073741823)
  -report #     set report interval (default: 1073741823)
  -seed #       set random seed (default: 1)
  -tenure #     set tabu tenure (default: 0)
  -time #       set cpu time limit in second (default: 600)

# = number, f = file name
```

と設定可能なオプションと解説が出力されます. これらのオプションは，以下のような意味をもちます（ここでは，制限時間を設定するオプション`-time` だけを使います）.

`-backtrack`

スケジュール生成 1 回あたりの最大バックトラック数を設定する.

`-data`

入力データを出力して終了する.

`-initial`

指定したファイルで初期解を与える.

`-iteration`

最大反復回数を設定する.

`-report`

解移動情報を何反復ごとに出力するか設定する.0 を指定すると, より詳しい情報が各反復ごとに出力される.

`-seed`

乱数系列の種を設定する.

`-tenure`

タブー期間 (tabu tenure) の初期値を設定する．タブー期間は探索中自動調節される．

`-time`

最大計算時間 (秒) を設定する．

制限時間を 1 秒として計算を実行するには，

`optseq -time 1<` *inputfile*

とします．

なお，結果を画面にではなく，ファイルに出力したい場合には，出力ファイル名を *outputfile* として，

`optseq <` *inputfile* `>` *outputfile*

とします．

この問題は極めて簡単なので，最適解が容易にみつかります．画面には，以下のような表示がされるはずです（結果の説明にコメントを追加してあります）．

```
# reading data ... done: 0.00(s) データを読むために必要な時間は０秒
# random seed: 1       乱数の初期値は１と設定
# tabu tenure: 1       タブーサーチのパラメータであるタブー期間の初期値は１
                       （内部で自動的にチューニングされる）
# cpu time limit: 3.00(s)  計算時間は３秒
# iteration limit: 1073741823 反復回数の上限
# computing all-pairs longest paths and strongly connected components ... done
       点間の最長路を計算し実行可能性を調べています
#scc 7
objective value = 55 (cpu time = 0.00(s), iteration = 0)
       目的関数値５５が CPU 時間０秒，反復０回で求まったことを表します
0: 0.00(s): 55/55
--- best activity list ---
source activity[2] activity[5] activity[1] activity[3] activity[4] sink 最適な作業の投入順序

--- best solution ---最良解
source ---: 0 0  ダミーの始点の開始時刻が０
sink ---: 55 55  ダミーの終点の開始時刻が５５（完了時刻）
activity[1] ---: 0 0--13 13 作業１は０に開始されて１３に終了
activity[2] ---: 0 0--25 25 作業２は０に開始されて２５に終了
activity[3] ---: 13 13--28 28 作業３は１３に開始されて２８に終了
activity[4] ---: 28 28--55 55 作業４は２８に開始されて５５に終了
activity[5] ---: 25 25--47 47 作業５は２５に開始されて４７に終了

objective value = 55 目的関数値は５５
cpu time = 0.00/3.00(s) 計算時間
iteration = 1/62605     反復回数
```

これは，最後の作業が完了する時間（離陸の時間）が 55 分後であり，そのための各作業の開始時間が，それぞれ $0, 0, 13, 28, 25$ 分後であることが分かります．

図 2 に，得られたスケジュールを図示します．この図式は，1919 年に Henry L. Gantt によって考案されたもので，ガントチャートとよばれます．ガントチャートでは，横軸を時間軸とし，作業時間を矩形で表示します．ガント

チャートを描くための **GUI**（Graphical User Interface）は，たくさん販売（もしくは無料配布）されていますので，OptSeq II に GUI を追加することは比較的容易です．Excel を用いても簡易的にガントチャートが描けます．より詳細な表示をするには，Microsoft Project などのソフトウェアもしくは市販の ActiveX コンポーネントをご利用ください．

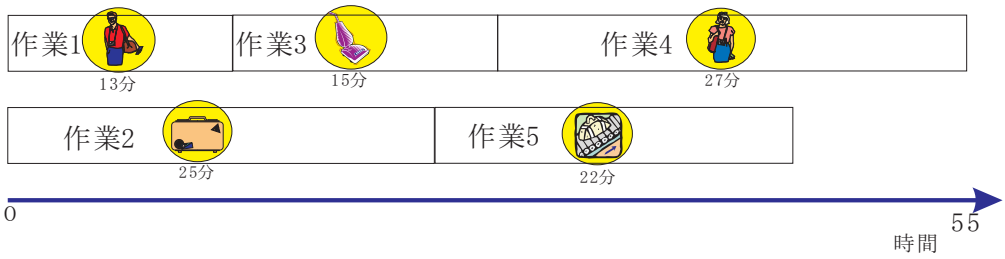

図 2: ガントチャート

入力データのすべてを以下にまとめておきます．

```
# PERT の例題
activity activity[1]
        mode duration 13

activity activity[2]
        mode  duration 25

activity activity[3]
        mode  duration 15

activity activity[4]
        mode  duration 27

activity activity[5]
        mode  duration 22

# 先行制約
temporal  activity[1] activity[3]
temporal  activity[2] activity[4]
temporal  activity[2] activity[5]
temporal  activity[3] activity[4]

# 最大完了時刻最小化
activity sink duedate 0
```

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう．

まず，OptSeq II のモジュールを読み込み，モデルオブジェクト `m1` を作成します．

```
from optseq2 import *
m1=Model()
```

次に，モデルに作業を追加します．そのために，作業時間を表すデータを，作業をキー作業時間を値とする辞書 `duration` として準備しておきます．次に，作業を保管するための空の辞書 `act` とモードを保管するための空の辞書 `mode` を作成します．

```
duration ={1:13, 2:25, 3:15, 4:27, 5:22 }
```

```
act={}
mode={}
```

次に，モデルオブジェクト m1 の addActivity メソッドを用いてモデルに作業を追加します．また，各作業にはモードを定義する必要があるので，モードクラス Mode を用いてモードに必要な作業時間を定義した後で，addModes メソッドを用いて作業にモードを追加します．

```
for i in duration:
    act[i]=m1.addActivity("Act[%s]"%i)
    mode[i]=Mode("Mode[%s]"%i,duration[i])
    act[i].addModes(mode[i])
```

次に，モデルに addTemporal メソッドを用いて先行制約を追加します．

```
m1.addTemporal(act[1],act[3])
m1.addTemporal(act[2],act[4])
m1.addTemporal(act[2],act[5])
m1.addTemporal(act[3],act[4])
```

最後に，m1.optimize() を入力し，実行すれば求解できます．求解の前にモデルのパラメターを変更することも可能です．

m1.Params.TimeLimit は求解するときの制限時間なので，それを 1 秒に設定しておきます．　また，m1.Params.OutputF
はモデルを出力する場合 True，出力しない場合 False を表すパラメータなので，それを True に設定しておきます．
m1.Params.Makespan は最大完了時刻最小化のときは True，それ以外のときは False を表すパラメータなので，それを True に設定しておきます．

```
m1.Params.TimeLimit=1
m1.Params.OutputFlag=True
m1.Params.Makespan=True
m1.optimize()
```

以下にすべての Python プログラムをまとめておきます．

```
from optseq2 import *

m1=Model()
duration ={1:13, 2:25, 3:15, 4:27, 5:22 }
act={}
mode={}
for i in duration:
    act[i]=m1.addActivity("Act[%s]"%i)
    mode[i]=Mode("Mode[%s]"%i,duration[i])
    act[i].addModes(mode[i])

#temporal (precedense) constraints
m1.addTemporal(act[1],act[3])
m1.addTemporal(act[2],act[4])
m1.addTemporal(act[2],act[5])
m1.addTemporal(act[3],act[4])

m1.Params.TimeLimit=1
m1.Params.OutputFlag=True
m1.Params.Makespan=True
m1.optimize()
```

ここで扱った例題は，久保幹雄著「組合せ最適化とアルゴリズム」（共立出版）から引用したものです．PERT に対する効率的なアルゴリズムについては，原著をご参照ください．

練習問題 1 自分で PERT の例題を作成して，それを OptSeq II を用いて解いてみましょう．

4 PERT（作業員が 1 人のとき）–資源を使ってみよう！–

ここで学ぶこと

- 資源制約の追加の仕方
- 作業の使用する資源量の入力の仕方

あなたは航空機会社のコンサルタントだ．リストラのため作業員の大幅な削減を迫られたあなたは，前節の例題と同じ問題を 1 人の作業員で行うためのスケジュールを作成しなければならなくなった．作業時間や先行制約は，前節と同じであるとするが，各々の作業は作業員を 1 人占有する（すなわち，2 つの作業を同時にできない）ものとする．どのような順序で作業を行えば，最短で離陸できるだろうか?

この例題のように，使用できる資源（作業員）に限りがある問題を扱うには，前節ではデータ設定をしていなかった資源をきちんと入力する必要があります．

資源とその使用可能量上限（供給量）は，以下のような書式で設定します．

```
resource 資源名
        interval 時刻 1   時刻 2   capacity 供給量
        interval 時刻 1   時刻 2   capacity 供給量
        ...
```

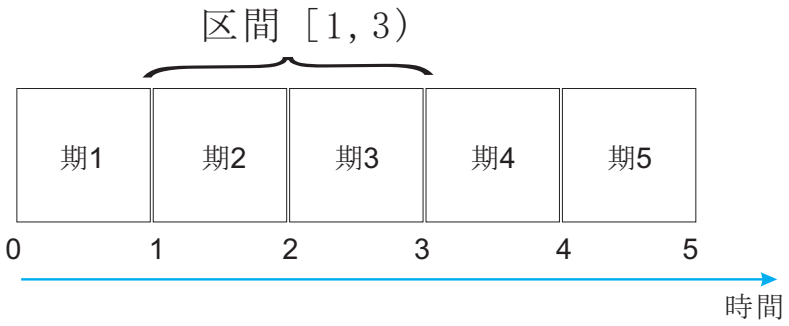

図 3: 区間の例

「時刻 1」は資源が使用できる開始時刻,「時刻 2」は終了時刻を表します．この時刻の組を**区間**（interval）とよびます．OptSeq II における区間は，[開始時刻 , 終了時刻) を表します．これは，開始時刻以上，終了時刻未満を表します．

離散的な時間を考えた場合には，時刻 $t-1$ から時刻 t の区間を**期** (period) t と定義します．時刻の初期値を 0 と仮定すると，期は 1 から始まる整数値をとることになります．区間 (開始時刻 , 終了時刻) に対応する期は,「開始時刻 +1，開始時刻 +2，$\cdots$，終了時刻」となります（図 3）．

区間はキーワード interval の後に記述し，1 つ以上，何個でも定義できます．区間ごとにキーワード capacity の後に，資源供給量を指定します．なお，記述のない区間は供給量 0 (資源使用不可) と見なされます．

例題においては，時刻 0 からすべての作業が完了するまで，資源量 1 を供給する作業員（名前は worker）を定義します．

```
resource worker  interval 0 inf  capacity 1
```

ここで，inf は無限大（infinity）を表し，非常に大きな数を表すキーワードです．

また，モード（この例題では作業と同じ意味です）に資源の使用量を追加する必要があります．そのためには，作業（もしくはモード）の定義の中で，資源量をキーワード requirement の後に記述します．

```
activity  作業名
  資源名  interval 区間  requirement  使用量
          interval 区間  requirement  使用量          ...
```

例題の場合には，各作業は作業員 1 人を占有するので，(たとえば）作業 1 の定義は，以下のようになります．

```
activity activity[1]
        mode duration 13
        worker interval 0 13 requirement 1
```

他の作業についても同様に資源使用量が 1 であることを記述します．
この問題も最適解が容易にみつかります．画面には，以下のような表示がされるはずです（図 4）．

```
--- best activity list ---
source activity[2] activity[5] activity[1] activity[3] activity[4] sink

--- best solution ---
source ---: 0 0
sink ---: 102 102
activity[1] ---: 47 47--60 60
activity[2] ---: 0 0--25 25
activity[3] ---: 60 60--75 75
activity[4] ---: 75 75--102 102
activity[5] ---: 25 25--47 47

objective value = 102
cpu time = 0.00/3.00(s)
iteration = 0/64983
```

図 4: 作業員が 1 人の場合のガントチャート

この結果は，最後の作業が完了する時間（離陸の時間）が 102 分後であり，そのための各作業の開始時間が，それぞれ $25, 0, 38, 53, 80$ 分後であることを表しています．
入力データのすべてを以下にまとめておきます．

```
# PERT の例題（作業員が 1 人の場合）

resource worker  interval 0 inf  capacity 1

activity activity[1]
        mode duration 13
        worker interval 0 13 requirement 1

activity activity[2]
        mode  duration 25
        worker interval 0 25 requirement 1

activity activity[3]
        mode  duration 15
        worker interval 0 15 requirement 1

activity activity[4]
        mode  duration 27
        worker interval 0 27 requirement 1

activity activity[5]
        mode  duration 22
        worker interval 0 22 requirement 1

# 先行制約
temporal  activity[1] activity[3]
temporal  activity[2] activity[4]
temporal  activity[2] activity[5]
temporal  activity[3] activity[4]

# 最大完了時刻最小化
activity sink duedate 0
```

注意！
作業時間が 0 のモードに対しては，資源を追加できません．しばしば高度なモデル化を行う場合に，作業時間が 0 のダミーの作業を使いますが，同時に資源を使用することはできませんのでご注意下さい．

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう．

資源（この例題の場合は作業員）`res` の使用可能量の上限（容量）は，モデルの `addResource` メソッドを用いて追加します．引数として（`"資源名",(開始時刻, 終了時刻):供給量`) を入力します．

```
res=m1.addResource("worker",capacity={(0,"inf"):1})
```

モードに資源の使用量を追加するときには，モードの `addResource` メソッドを用いて追加します．引数として（資源,(開始時刻, 終了時刻):使用量) を入力します．ここでは区間を，各作業の作業時間（この例題の場合作業 1 は 0 から 13 ，作業 2 は 0 から 25 など）と書かずに `0,"inf"`と書いてあります．これはモードに作業時間（この例題の場合 3 行目の `duration[i]`）) を入力してあるため，この作業時間外では資源が使われないからです．

```
1 for i in duration:
2     act[i]=m1.addActivity("Act[%s]"%i)
```

```
3       mode[i]=Mode("Mode[%s]"%i,duration[i])
4       mode[i].addResource(res,{(0,"inf"):1})
5       act[i].addModes(mode[i])
```

以下にすべての Python プログラムをまとめておきます.

```
from optseq2 import *

m1=Model()
duration ={1:13, 2:25, 3:15, 4:27, 5:22 }
res=m1.addResource("worker",capacity={(0,"inf"):1})
act={}
mode={}
for i in duration:
    act[i]=m1.addActivity("Act[%s]"%i)
    mode[i]=Mode("Mode[%s]"%i,duration[i])
    mode[i].addResource(res,{(0,"inf"):1})
    act[i].addModes(mode[i])

#temporal (precedense) constraints
m1.addTemporal(act[1],act[3])
m1.addTemporal(act[2],act[4])
m1.addTemporal(act[2],act[5])
m1.addTemporal(act[3],act[4])

print m1
m1.Params.TimeLimit=1
m1.Params.OutputFlag=True
m1.Params.Makespan=True
m1.optimize()
```

練習問題 2 前節の練習問題で作成した PERT の例題に対して，作業員 1 人で作業を行うとしたときのスケジュールを OptSeq II を用いて解いてみましょう.

5 並列ショップスケジューリング–ピットイン時間を短縮せよ!–

ここで学ぶこと

- 並列ショップスケジューリングについて
- 資源量の上限が 1 より大きい場合の取り扱い

あなたは F1 のピットクルーだ．F1 レースにとってピットインの時間は貴重であり，ピットインしたレーシングカーに適切な作業を迅速に行い，なるべく早くレースに戻してやることが，あなたの使命である．

作業 1：給油準備 (3 秒)

作業 2：飲料水の取り替え　(2 秒)

作業 3：フロントガラス拭き (2 秒)

作業 4：ジャッキで車を持ち上げ (2 秒)

作業 5：タイヤ (前輪左側) 交換 (4 秒)

作業 6：タイヤ (前輪右側) 交換 (4 秒)

作業 7：タイヤ (後輪左側) 交換 (4 秒)

作業 8：タイヤ (後輪右側) 交換 (4 秒)

作業 9：給油 (11 秒)

作業 10：ジャッキ降ろし (2 秒)

各作業には，作業時間のほかに，この作業が終わらないと次の作業ができないといったような先行制約がある．作業時間と先行制約は，図 5 のようになっている．
いま，あなたを含めて 3 人のピットクルーがいて，これらの作業を手分けして行うものとする．作業は途中で中断できないものとすると，なるべく早く最後の作業を完了させるには，誰がどの作業をどういう順番で行えばよいのだろうか?

この問題は，**並列ショップ**（parallel shop）スケジューリングとよばれる問題です．

3 人の作業員（ピットクルー）が同一である（区別しない）と考えたときには，資源量上限が 3 の資源が 1 つあるものとしてモデル化できます．

```
# 並列ショップスケジューリングの例題（ピットイン時間を短縮せよ！）

resource worker  interval 0 inf  capacity 3

activity prepare
        mode duration 3
        worker interval 0 3 requirement 1

activity water
        mode  duration 2
        worker interval 0 2 requirement 1

activity front
        mode  duration 2
        worker interval 0 2 requirement 1

activity jackup
        mode  duration 2
        worker interval 0 2 requirement 1

activity tire1
        mode  duration 4
```

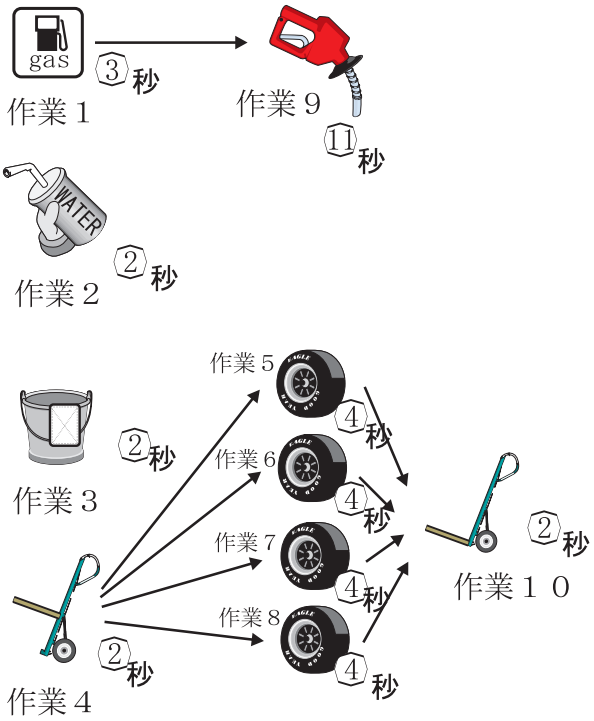

図 5: ピットクルーの作業の作業時間と先行制約

```
        worker interval 0 4 requirement 1

activity tire2
        mode  duration 4
        worker interval 0 4 requirement 1

activity tire3
        mode  duration 4
        worker interval 0 4 requirement 1

activity tire4
        mode  duration 4
        worker interval 0 4 requirement 1

activity oil
        mode  duration 11
        worker interval 0 11 requirement 1

activity jackdown
        mode  duration 2
        worker interval 0 2 requirement 1

# 先行制約
temporal  prepare oil
temporal  jackup tire1
temporal  jackup tire2
temporal  jackup tire3
temporal  jackup tire4
temporal  tire1 jackdown
temporal  tire2 jackdown
temporal  tire3 jackdown
```

```
temporal  tire4 jackdown

# 最大完了時刻最小化
activity sink duedate 0
```

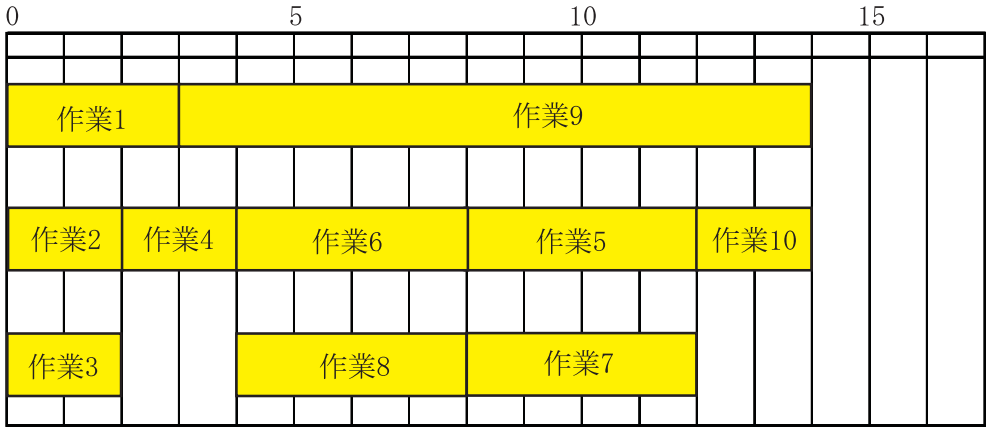

図 6: ピットイン時間 14 秒のスケジュール（ガントチャート）

OptSeq II を実行すると，以下の結果が得られます．

```
--- best activity list ---
source prepare water oil front jackup tire4 tire2 tire1 tire3 jackdown sink

--- best solution ---
source ---: 0 0
sink ---: 14 14
prepare ---: 0 0--3 3
water ---: 0 0--2 2
front ---: 0 0--2 2
jackup ---: 2 2--4 4
tire1 ---: 8 8--12 12
tire2 ---: 4 4--8 8
tire3 ---: 8 8--12 12
tire4 ---: 4 4--8 8
oil ---: 3 3--14 14
jackdown ---: 12 12--14 14

objective value = 14
cpu time = 0.00/3.00(s)
iteration = 0/37644
```

これは完了時刻 14 のスケジュールで，作業 1 の後に作業 9 を行わなければならないことを考えると最適であることがわかります．得られたスケジュールをガントチャートで図示すると，図 6 のようになります．

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう．

この例題は， 3 人の作業員がいるので，資源 `res` に関する制約を追加するとき，引数（"資源名",(開始時刻, 終了時刻):供給量) の供給量を 3 に設定します．

```
res=m1.addResource("worker",capacity={(0,"inf"):3})
```

以下にすべての Python プログラムをまとめておきます．

```
  activity    mode   duration | 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|
-------------------------------------------------------------------------|
 Act[10]   Mode[10]     2     |  |  |  |  |  |  |  |  |  |  |  |  |==|==|
  Act[1]   Mode[1]      3     |==|==|==|  |  |  |  |  |  |  |  |  |  |  |
  Act[2]   Mode[2]      2     |==|==|  |  |  |  |  |  |  |  |  |  |  |  |
  Act[3]   Mode[3]      2     |==|==|  |  |  |  |  |  |  |  |  |  |  |  |
  Act[4]   Mode[4]      2     |  |  |==|==|  |  |  |  |  |  |  |  |  |  |
  Act[5]   Mode[5]      4     |  |  |  |  |  |  |  |  |==|==|==|==|  |  |
  Act[6]   Mode[6]      4     |  |  |  |  |==|==|==|==|  |  |  |  |  |  |
  Act[7]   Mode[7]      4     |  |  |  |  |  |  |  |  |==|==|==|==|  |  |
  Act[8]   Mode[8]      4     |  |  |  |  |==|==|==|==|  |  |  |  |  |  |
  Act[9]   Mode[9]      11    |  |  |  |==|==|==|==|==|==|==|==|==|==|==|
-------------------------------------------------------------------------
   resource usage/capacity    |
------------------------------------------------------------------------
            worker            | 3| 3| 2| 2| 3| 3| 3| 3| 3| 3| 3| 2| 2|
                              | 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3|
------------------------------------------------------------------------
```

図 7: 例題の簡易ガントチャート表示．==は活動を処理中を表します．

```
from optseq2 import *

m1=Model()
duration ={1:3, 2:2, 3:2, 4:2, 5:4, 6:4, 7:4, 8:4, 9:11, 10:2 }
res=m1.addResource("worker",capacity={(0,"inf"):3})
act={}
mode={}
for i in duration:
    act[i]=m1.addActivity("Act[%s]"%i)
    mode[i]=Mode("Mode[%s]"%i,duration[i])
    mode[i].addResource(res,{(0,"inf"):1})
    act[i].addModes(mode[i])

#temporal (precedense) constraints
m1.addTemporal(act[1],act[9])
for i in range(5,9):
    m1.addTemporal(act[4],act[i])
    m1.addTemporal(act[i],act[10])

print m1
m1.Params.TimeLimit=1
m1.Params.OutputFlag=True
m1.Params.Makespan=True
m1.optimize()
m1.write("chart3.txt")
```

上のプログラムでは，最適化を行った後に，`m1.write("ファイル名.txt")` を入力することによって，簡易ガントチャートを書き出しています．簡易ガントチャートの出力結果は，以下のようになります．

また，OptSeq の Python インターフェイスでは，Excel に入力するためのカンマ区切りのテキストファイルを出力することも可能です．そのためには，`m1.writeExcel("ファイル名.csv")` という書式を用います．Excel には，入力可能な列数に制限があるので，注意してお使い下さい．

ここで扱った例題は，久保幹雄，松井知己著「組合せ最適化短編集」（朝倉書店）から引用したものです．そこでは，通常の工場内で用いられている**ディスパッチング・ルール**（dispatching rule）のような単純な（処理的な）ヒューリスティクスを用いると，直感に合わない変な結果が出てくる可能性があることを，お話を通して解説しています．詳

細については，原著をご参照ください．

練習問題 3 上の例題を，作業員 4 人で作業を行うとしたときのスケジュールを OptSeq II を用いて解いてみましょう．また，得られたスケジュールをディスパッチング・ルールのようなヒューリスティクスで解いた場合のスケジュールと比べてみましょう．

練習問題 4 上の例題を，先行制約をなくしたと仮定して，OptSeq II を用いて解いてみましょう．また，得られたスケジュールをディスパッチング・ルールのようなヒューリスティクスで解いた場合のスケジュールと比べてみましょう．

練習問題 5 上の例題を，作業時間をすべて 1 秒短縮したと仮定して，OptSeq II を用いて解いてみましょう．また，得られたスケジュールをディスパッチング・ルールのようなヒューリスティクスで解いた場合のスケジュールと比べてみましょう．

6 並列ショップスケジューリング 2–モードの概念と使用法–

ここで学ぶこと

- モードの概念と使用法
- モードの入力の仕方

ここでは，前節の例題を「モード」の概念を用いて解いてみましょう．まず，モードとは何か，ならびにその使用方法について解説します．

1 つの作業でも，複数の異なった処理方法をとることが可能な場合が，実際問題では多くあります．たとえば,「ジュースを買う」という作業を行うときにも,「コンビニに買いに行く」,「自販機で買う」,「スーパーで割り引き中のジュースを買いに行く」など，様々な処理方法が考えられます．OptSeq II では，このように作業の処理の仕方が複数存在する場合は，作業に「モード」を付加することによって解決します．作業は，与えられたいずれか 1 つのモードを選択することによって処理されます．モードごとに，作業時間，使用する資源とその使用量などを設定できます．たとえば,「ジュースを買う」という作業の例では，3 つのモードを設定し，モードごとのデータを

コンビニモード： 作業時間 20 分，お金資源 120 円，人資源 1 人

自販機モード： 作業時間 5 分，お金資源 120 円，人資源 1 人

スーパーモード： 作業時間 40 分，お金資源 100 円，人資源 1 人，車資源 1 台

のように設定します．

また，モードは作業を行う人数（や処理を行う機械の台数）によって作業時間が短縮される場合にもよく使われます．

前節で扱った 3 人の作業員が,「給油準備作業」を協力して作業を行い，時間短縮ができる場合を考えます．1 人でやれば 3 秒かかる作業が，2 人でやれば 2 秒，3 人がかりなら 1 秒で終わるものとします．これは，作業に 3 つのモードをもたせ，それぞれ作業時間と使用資源量を以下のように設定することによって表現できます．

モード 1： 作業時間 3 秒，人資源 1 人

モード 2： 作業時間 2 秒，人資源 2 人

モード 3： 作業時間 1 秒，人資源 3 人

1 つの作業が複数のモードをもつ場合には，モードの作業時間と必要な資源量を，{ の中に記述して表すことにします（図 8）.

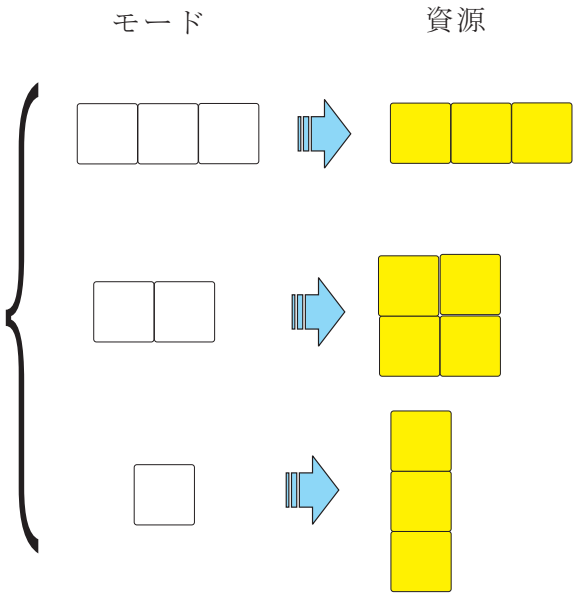

図 8: 3 種類のモードの表現

これを記述するためには，まず，3 つのモードをキーワード mode で記述します．モードの記述方法は，作業（activity）と同じで，

```
mode  処理モード名  duration  処理時間
  資源名  interval 区間  requirement  使用量
          interval 区間  requirement  使用量          ...
```

と定義します.

例題のデータでは，

```
mode m1 duration 3
        worker interval 0 3 requirement 1
```

```
mode m2 duration 2
        worker interval 0 2 requirement 2

mode m3 duration 1
        worker interval 0 1 requirement 3
```

となります.

その後で，作業にモードを追加します．書式は以下の通りです.

```
activity  作業名
   モード名    モード名       ...
```

例題における給油準備を表す作業（名称は prepare）に，上で定義した 3 つのモード（名称は m1, m2, m3）を追加します.

```
activity prepare
        m1   m2  m3
```

作業時間はモードで記述したので，作業の定義内では書く必要はありません.

入力データ全体を以下に示しておきます.

```
# 並列ショップスケジューリングの例題（モードを用いた場合）

resource worker  interval 0 inf  capacity 3

mode m1 duration 3
        worker interval 0 3 requirement 1

mode m2 duration 2
        worker interval 0 2 requirement 2

mode m3 duration 1
        worker interval 0 1 requirement 3

activity prepare
        m1   m2  m3

activity water
        mode  duration 2
        worker interval 0 2 requirement 1

activity front
        mode  duration 2
        worker interval 0 2 requirement 1

activity jackup
        mode  duration 2
        worker interval 0 2 requirement 1
```

```
activity tire1
        mode  duration 4
        worker interval 0 4 requirement 1

activity tire2
        mode  duration 4
        worker interval 0 4 requirement 1

activity tire3
        mode  duration 4
        worker interval 0 4 requirement 1

activity tire4
        mode  duration 4
        worker interval 0 4 requirement 1

activity oil
        mode  duration 11
        worker interval 0 11 requirement 1

activity jackdown
        mode  duration 2
        worker interval 0 2 requirement 1

# 先行制約
temporal  prepare oil
temporal  jackup tire1
temporal  jackup tire2
temporal  jackup tire3
temporal  jackup tire4
temporal  tire1 jackdown
temporal  tire2 jackdown
temporal  tire3 jackdown
temporal  tire4 jackdown

# 最大完了時刻最小化
activity sink duedate 0
```

結果は，以下のようになります.

```
--- best activity list ---
source prepare oil jackup tire4 tire2 tire1 water tire3 front jackdown sink

--- best solution ---
source ---: 0 0
sink ---: 13 13
prepare m3: 0 0--1 1
water ---: 1 1--3 3
front ---: 11 11--13 13
jackup ---: 1 1--3 3
tire1 ---: 7 7--11 11
tire2 ---: 3 3--7 7
tire3 ---: 7 7--11 11
tire4 ---: 3 3--7 7
oil ---: 1 1--12 12
jackdown ---: 11 11--13 13

objective value = 13
cpu time = 0.00/3.00(s)
iteration = 7/23318
```

結果は，給油準備作業（prepare）を 3 人の作業員が共同で行うモード（モード m3）で行うことによって，1 秒短縮した 13 秒で終わることが分かります（図 9）.

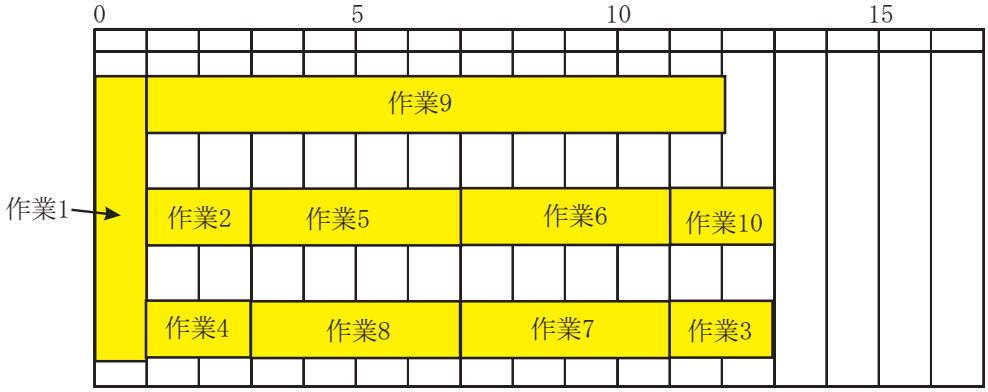

図 9: 給油準備作業を 3 人が協力して行ったピットイン時間 13 秒のスケジュール

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう.

Python では，多モードの場合と単一モードの場合，モードの追加と資源の追加は同じ仕様になります．この例題では，作業 1 に 3 種類のモードがあると設定しています．それを Python で入力すると以下のようになります.（ここで `mode` は辞書ですので作業 1 のモード 1 は `mode[1,1]` と書きます.）

```
act={}
mode={}
for i in duration:
    act[i]=m1.addActivity("Act[%s]"%i)
    if i==1:
        mode[1,1]=Mode("Mode[1_1]",3)
        mode[1,1].addResource(res,{(0,"inf"):1})
        mode[1,2]=Mode("Mode[1_2]",2)
        mode[1,2].addResource(res,{(0,"inf"):2})
        mode[1,3]=Mode("Mode[1_3]",1)
        mode[1,3].addResource(res,{(0,"inf"):3})
        act[i].addModes(mode[1,1],mode[1,2],mode[1,3])
    else:
        mode[i]=Mode("Mode[%s]"%i,duration[i])
        mode[i].addResource(res,{(0,"inf"):1})
        act[i].addModes(mode[i])
```

以下にすべての Python プログラムをまとめておきます.

```
from optseq2 import *

m1=Model()
duration ={1:3, 2:2, 3:2, 4:2, 5:4, 6:4, 7:4, 8:4, 9:11, 10:2 }
res=m1.addResource("worker",capacity={(0,"inf"):3})
act={}
mode={}

for i in duration:
    act[i]=m1.addActivity("Act[%s]"%i)
    if i==1:
        mode[1,1]=Mode("Mode[1_1]",3)
        mode[1,1].addResource(res,{(0,"inf"):1})
        mode[1,2]=Mode("Mode[1_2]",2)
        mode[1,2].addResource(res,{(0,"inf"):2})
        mode[1,3]=Mode("Mode[1_3]",1)
        mode[1,3].addResource(res,{(0,"inf"):3})
```

```
  activity    mode   duration | 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|
-------------------------------------------------------------------------|
 Act[10]   Mode[10]     2     |  |  |  |  |  |  |  |  |  |  |==|==|
  Act[1]  Mode[1_3]     1     |==|  |  |  |  |  |  |  |  |  |  |  |
  Act[2]   Mode[2]      2     |  |==|==|  |  |  |  |  |  |  |  |  |
  Act[3]   Mode[3]      2     |  |  |  |  |  |  |  |  |  |  |==|==|
  Act[4]   Mode[4]      2     |  |==|==|  |  |  |  |  |  |  |  |  |
  Act[5]   Mode[5]      4     |  |  |  |  |  |  |  |==|==|==|==|  |
  Act[6]   Mode[6]      4     |  |  |  |==|==|==|==|  |  |  |  |  |
  Act[7]   Mode[7]      4     |  |  |  |  |  |  |  |==|==|==|==|  |
  Act[8]   Mode[8]      4     |  |  |  |==|==|==|==|  |  |  |  |  |
  Act[9]   Mode[9]      11    |  |==|==|==|==|==|==|==|==|==|==|  |
-------------------------------------------------------------------
   resource usage/capacity    |
-------------------------------------------------------------------
            worker            | 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 2|
                              | 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3|
-------------------------------------------------------------------
```

図 10: 例題の簡易ガントチャート表示．==は活動を処理中を表します．

```
        act[i].addModes(mode[1,1],mode[1,2],mode[1,3])
    else:
        mode[i]=Mode("Mode[%s]"%i,duration[i])
        mode[i].addResource(res,{(0,"inf"):1})
        act[i].addModes(mode[i])

#temporal (precedense) constraints
m1.addTemporal(act[1],act[9])
for i in range(5,9):
    m1.addTemporal(act[4],act[i])
    m1.addTemporal(act[i],act[10])

print m1
m1.Params.TimeLimit=1
m1.Params.OutputFlag=True
m1.Params.Makespan=True
m1.optimize()
m1.write("chart4.txt")
```

上のプログラムでも最適化を行った後に，`m1.write("ファイル名")` を入力することによって，簡易ガントチャートを書き出しています．簡易ガントチャートの出力結果は，以下のようになります．

7　資源制約付きスケジューリング–お家を早く造ろう!–

ここで学ぶこと

- 時間によって変化する資源量上限の入力の仕方
- 作業開始からの経過時間によって変化する資源の必要量の入力の仕方

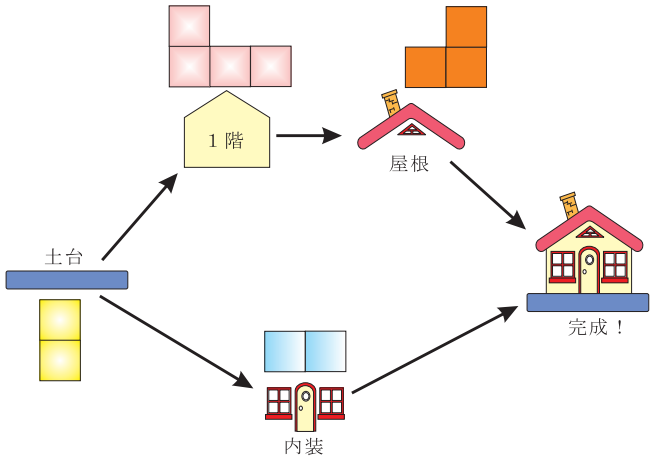

図 11: お家を造るための作業，先行制約，必要な人数

あなたは 1 階建てのお家を造ろうとしている大工さんだ．あなたの仕事は，なるべく早くお家を完成させることだ．お家を造るためには，幾つかの作業をこなさなければならない．まず，土台を造り，1 階の壁を組み立て，屋根を取り付け，さらに 1 階の内装をしなければならない．ただし，土台を造る終える前に 1 階の建設は開始できず，内装工事も開始できない．また，1 階の壁を作り終える前に屋根の取り付けは開始できない．
各作業とそれを行うのに必要な時間（単位は日）は，以下のようになっている．

土台：2 人の作業員で 1 日

1 階の壁：最初の 1 日目は 2 人，その後の 2 日間は 1 人で，合計 3 日

内装：1 人の作業員で 2 日

屋根：最初の 1 日は 1 人，次の 1 日は 2 人の作業員が必要で，合計 2 日

いま，作業をする人は，あなたをあわせて 2 人いるが，相棒の 1 人は作業開始 3 日目に休暇をとっている．
さて，最短で何日でお家を造ることができるだろうか?

作業間の先行制約と作業に必要な時間ならびに人数は，図 11 のようになっています．この図には，すべての作業が終了したあとを表すダミーの作業（お家の完成！）が追加されています．この完成を表す作業をなるべく早く終了させることが，問題の目的となります．

この例題を OptSeq II で解いてみましょう．

まず，資源に関するデータを入力しますが，問題になるのは作業員の内の 1 人が 3 日目に休暇をとっていることです．

この例題における休暇を表現するには，0 日目から 2 日目まで 2 人，2 日目から 3 日目まで 1 人，3 日目から最後まで 2 人と入れます．

```
resource worker
  interval 0 2    capacity 2
```

```
  interval 2 3    capacity 1
  interval 3 inf  capacity 2
```

作業とモードの入力の仕方は，前節までの例題と同じです．

さて，1 階の壁の取り付け作業は，最初の 1 日目は 2 人，その後の 2 日間は 1 人で，合計 3 日間かかりました．このように日ごとに変化する資源使用量（人数）を入力するには，複数の区間に対して，資源の使用量を入力します．

```
activity first
        mode  duration 3
        worker interval 0 1 requirement 2
        worker interval 1 3 requirement 1
```

他の部分の入力は，前節までと同様です．念のためすべての入力データを以下に示します．

```
# 資源制約付きスケジューリング--お家を早く造ろう!--

resource worker
  interval 0 2    capacity 2
  interval 2 3    capacity 1
  interval 3 inf  capacity 2

activity base
        mode  duration 1
        worker interval 0 1 requirement 2

activity first
        mode  duration 3
        worker interval 0 1 requirement 2
        worker interval 1 3 requirement 1

activity interior
        mode  duration 2
        worker interval 0 2 requirement 1

activity ceil
        mode  duration 2
        worker interval 0 1 requirement 1
        worker interval 1 2 requirement 2

# 先行制約
temporal  base first
temporal  base interior
temporal  first ceil

# 最大完了時刻最小化
activity sink duedate 0
```

OptSeq II によって求解すると，以下のような結果が得られます．

```
--- best activity list ---
source base first interior ceil sink

--- best solution ---
source ---: 0 0
sink ---: 6 6
```

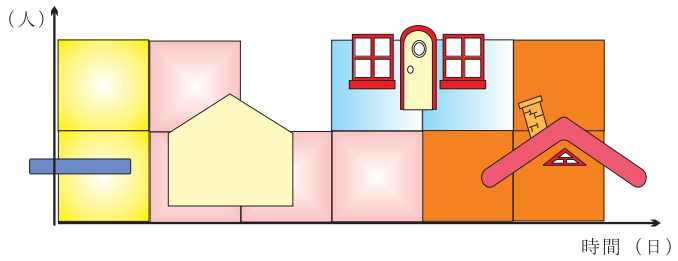

図 12: お家を造るためのスケジュール（ガントチャート）

```
base ---: 0 0--1 1
first ---: 1 1--4 4
interior ---: 3 3--5 5
ceil ---: 4 4--6 6

objective value = 6
cpu time = 0.00/3.00(s)
iteration = 1/38887
```

これは，土台は 1 日目，1 階は 2 日目，内装は 4 日目，屋根は 5 日目に作業を開始するスケジュールを表しています．完了時刻は 6 日で，資源の制約がない場合（PERT の場合）の完了時刻と一致することから，最適解であることがわかります．図 12 に得られたスケジュールと使用する資源量を示します．

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう．

資源使用量は，空の辞書 `req` を作り（ 1 行目），各作業の作業時間と資源の使用量を辞書 `req` に取っておきます．例えば， 2 行目の `req[1]=(0,1):2` は，作業 1 が，時刻 0 から時刻 1 （ 1 日目）に資源を（作業員） 2 単位使うことを表します．

```
1 req={}
2 req[1]={(0,1):2 }
3 req[2]={(0,1):2 ,(1,3):1}
4 req[3]={(0,2):1 }
5 req[4]={(0,1):1,(1,2):2 }
```

また，使用できる資源量も日によって変わりますので以下のように入力します．まず，モデルの `addResource` メソッドを用いて，資源名を`"worker"`とする資源 `res` を作ります．次に，資源の `addCapacity` メソッドを用いて資源制約を入力します．例えば， 2 行目の `res.addCapacity(0,2,2)` は，時刻 0 から時刻 2 （ 1 日目と 2 日目）に資源が 2 単位使えることを表します．

```
1 res=m1.addResource("worker")
2 res.addCapacity(0,2,2)
3 res.addCapacity(2,3,1)
4 res.addCapacity(3,"inf",2)
```

作業とモードの追加方法は，前節までの例題と同じです．

以下にすべての Python プログラムをまとめておきます．

```
from optseq2 import *

m1=Model()
duration ={1:1,2:3,3:2,4:2}
```

```
  activity    mode   duration |1|2|3|4|5|6|
-----------------------------------------|
  Act[1]   Mode[1]      1     |=| | | | | |
  Act[2]   Mode[2]      3     | |=|=|=| | |
  Act[3]   Mode[3]      2     | | | |=|=| |
  Act[4]   Mode[4]      2     | | | | |=|=|
-----------------------------------------
   resource usage/capacity    |
-----------------------------------------
            worker            |2|2|1|2|2|2|
                              |2|2|1|2|2|2|
-----------------------------------------
```

図 13: 例題の簡易ガントチャート表示．==は活動を処理中を表します．

```
req={}
req[1]={(0,1):2 }
req[2]={(0,1):2 ,(1,3):1}
req[3]={(0,2):1 }
req[4]={(0,1):1,(1,2):2 }
res=m1.addResource("worker")
res.addCapacity(0,2,2)
res.addCapacity(2,3,1)
res.addCapacity(3,"inf",2)

act={}
mode={}
for i in duration:
    act[i]=m1.addActivity("Act[%s]"%i)
    mode[i]=Mode("Mode[%s]"%i,duration[i])
    mode[i].addResource(res,req[i])
    act[i].addModes(mode[i])

#temporal (precedense) constraints
m1.addTemporal(act[1],act[2])
m1.addTemporal(act[1],act[3])
m1.addTemporal(act[2],act[4])

print m1
m1.Params.TimeLimit=1
m1.Params.OutputFlag=True
m1.Params.Makespan=True
m1.optimize()
m1.write("chart5.txt")
```

上のプログラムでも最適化を行った後に，`m1.write("ファイル名")` を入力することによって，簡易ガントチャートを書き出しています．簡易ガントチャートの出力結果は，以下のようになります．

練習問題 6 2 階建てのお家を造る問題を考え，それを OptSeq II を用いて解いてみましょう．

8 納期遅れを最小にしよう!

ここで学ぶこと

- 作業ごとに異なる納期の設定法
- 重み付きの納期遅れの利用法

あなたは売れっ子連載作家だ．あなたは，A, B, C, D の 4 社から原稿を依頼されており，それぞれ，どんなに急いで書いても 1 日，2 日，3 日，4 日かかるものと思われる．各社に約束した納期は，それぞれ 5 日後，9 日後，9 日後，4 日後であり，納期から 1 日遅れるごとに 1 万円の遅延ペネルティを払わなければならない．

会社名	A	B	C	D
作業時間（日）	1	2	3	4
納期（日後）	5	9	6	4

どのような順番で原稿を書けば，支払うペネルティ料の合計を最小にできるだろうか?

上の例題は，納期遅れ最小化を目的とした 1 機械スケジューリングとよばれる問題です．OptSeq II では，作業名の後ろに，キーワード duedate を付加することによって，作業の納期を設定できます．

```
activity  作業名 duedate 納期 weight 重み
```

重み（weight）は納期遅れに対するペナルティを表します．納期遅れをなるべくしたくない作業に対しては，重みを大きく設定します．重みを省略した場合には，1 の重みが自動的に付加されます．

たとえば，作業 A （名称は A）の納期を 5 日後に設定するためには，以下のように入力します．

```
activity A duedate 5
```

納期遅れのペナルティが，1 万円ではなく，10 万円の場合には，以下のようになります．

```
activity A duedate 5 weight 10
```

以下に入力データの全体を示します．

```
# 納期遅れを最小にしよう!

resource writer
  interval 0 inf  capacity 1

activity A duedate 5
        mode  duration 1
        writer interval 0 1 requirement 1

activity B duedate 9
        mode  duration 2
        writer interval 0 2 requirement 1

activity C duedate 6
        mode  duration 3
        writer interval 0 3 requirement 1

activity D duedate 4
        mode  duration 4
        writer interval 0 4 requirement 1
```

結果は，以下のようになります．

```
--- best activity list ---
source D A C B sink

--- best solution ---
source ---: 0 0
sink ---: 10 10
A ---: 4 4--5 5
B ---: 8 8--10 10
C ---: 5 5--8 8
D ---: 0 0--4 4

objective value = 3
cpu time = 0.00/3.00(s)
iteration = 0/57376
```

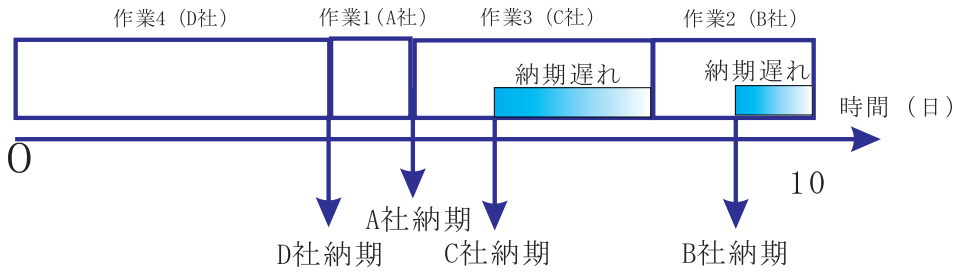

図 14: 1 機械スケジューリング問題の結果（ガントチャート）

これは，D,A,C,B 社の順に仕事をすれば良いことを表しています．納期遅れは，C 社に対して 2 日，B 社に対して 1 日で，合計 3 日分（3 万円）のペナルティを支払えば良いことを表します（図 14）．

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう．

まず，各作業の納期を，作業番号をキー，納期を値とした辞書として準備しておきます．

```
due={1:5,2:9,3:6,4:4}
```

次に，納期と納期遅れのペナルティを作業に追加します．納期と納期遅れのペナルティは，モデルに作業を追加するとき，`addActivity("作業名",duedate=納期,weight=ペナルティ)` のように入力します．また，ペナルティの規定値は 1 ですので，この例題の場合，ペナルティの入力を省略しました（プログラムの 2 行目）．

```
1 for i in req:
2     act[i]=m1.addActivity("Act[%s]"%i,duedate=due[i])
3     mode[i]=Mode("Mode[%s]"%i,req[i])
4     mode[i].addResource(res,{(0,"inf"):1})
5     act[i].addModes(mode[i])
```

この例題の場合，納期遅れ最小化を目的とするため，モデルのパラメータ `Makespan` を `False` に設定します．(ただしパラメータ `Makespan` の規定値は `False` であるので省略してもかまいません．)

```
m1.Params.Makespan=False
```

以下にすべての Python プログラムをまとめておきます．

```
from optseq2 import *

m1=Model()
```

```
  activity    mode   duration | 1| 2| 3| 4| 5| 6| 7| 8| 9|10|
-------------------------------------------------------------|
  Act[1]   Mode[1]      1     |  |  |  |  |==|  |  |  |  |  |
  Act[2]   Mode[2]      2     |  |  |  |  |  |  |  |  |==|==|
  Act[3]   Mode[3]      3     |  |  |  |  |  |==|==|==|  |  |
  Act[4]   Mode[4]      4     |==|==|==|==|  |  |  |  |  |  |
-------------------------------------------------------------
   resource usage/capacity    |
-------------------------------------------------------------
            writer            | 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
                              | 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
-------------------------------------------------------------
```

図 15: 例題の簡易ガントチャート表示．==は活動を処理中を表します．

```
due={1:5,2:9,3:6,4:4}
req={1:1, 2:2, 3:3, 4:4 }
res=m1.addResource("writer")
res.addCapacity(0,"inf",1)

act={}
mode={}
for i in req:
    act[i]=m1.addActivity("Act[%s]"%i,duedate=due[i])
    mode[i]=Mode("Mode[%s]"%i,req[i])
    mode[i].addResource(res,{(0,"inf"):1})
    act[i].addModes(mode[i])

print m1
m1.Params.TimeLimit=1
m1.Params.OutputFlag=True
m1.Params.Makespan=False
m1.optimize()
m1.write("chart6.txt")
```

上のプログラムでも最適化を行った後に，`m1.write("ファイル名")` を入力することによって，簡易ガントチャートを書き出しています．簡易ガントチャートの出力結果は，以下のようになります．

9　航空機をもっと早く離陸させよう!（CPM）

ここで学ぶこと

- 再生不能資源（お金や原材料など使用するとなくなってしまうタイプの資源）の表現法
- クリティカルパス法（CPM）のモードと再生不能資源による表現法

あなたは，§3 や §4 で登場した航空機会社のコンサルタントだ．あなたは，以前と同じ設定で，作業時間の短縮を要求されている．(ただし，§3 と同様に，資源制約（人の制限）はないものとする．) いま，航空機の離陸の前にする作業の時間が，費用をかけることによって短縮でき，各作業の標準時間，新設備導入によって短縮したときの時間，ならびにそのときに必要な費用は，以下のように推定されているものとする．

作業 1：乗客降ろし 13 分．10 分に短縮可能で，1 万円必要．

作業 2：荷物降ろし 25 分．20 分に短縮可能で，1 万円必要．

作業 3：機内清掃 15 分．10 分に短縮可能で， 1 万円必要．

作業 4：新しい乗客の搭乗 27 分．25 分に短縮可能で， 1 万円必要．

作業 5：新しい荷物積み込み 22 分．20 分に短縮可能で， 1 万円必要．

さて，いくら費用をかけると，どのくらい離陸時刻を短縮するすることができるだろうか?

これは，**クリティカルパス法**（critical path method，略して CPM）とよばれる古典的な問題です．CPM は，作業時間を費用（お金）をかけることによって短縮できるという仮定のもとで，費用と作業完了時刻のトレードオフ曲線を求めることを目的とした PERT の変形で，資源制約がないときには効率的な解法が古くから知られていますが，資源制約がつくと困難な問題になります．ここでは，この問題が,「モード」と「再生不能資源」を用いて，OptSeq II で簡単にモデリングできることを示します．

さて，作業は通常の作業時間と短縮時の作業時間をもちますが，これは作業に付随するモードで表現することにしましょう．問題となるのは，作業時間を短縮したときには，お金がかかることです．お金は資源の一種と考えられますが，いままで考えていた資源とは若干異なります．

いままで考えていた資源は，機械や人のように，作業時間中は使用されていますが，作業が終了すると，再び別の作業で使うことができるようになります．このように，作業完了後に再び使用可能になる資源を,「再生可能資源」とよびます．

一方，お金や原材料のように，一度使うとなくなってしまう資源も考えられます．このような資源を，**再生不能資源**（nonrenewable resource）とよびます．

例題に対して，再生不能資源（お金）の上限を色々変えて最短時間を求めてみましょう．まず，各々の作業に対して，通常の作業時間をもつ場合と，短縮された作業時間をもつ場合の 2 つのモードを追加します．以下に，作業 1（乗客降ろし）に対して，13 分の通常モードと，10 分の短縮モードを追加するための入力を示します．

```
mode m[1][1] duration 13
mode m[1][2] duration 10


activity activity[1]
        m[1][1] m[1][2]
```

他の 4 つの作業に対しても上と同様に 2 つずつモードを追加します．

次に，2 番目のモード（名称は m[1..5][2]）を用いたときに，再生不能資源（お金）を 1 万円かかることを記述します．

再生不能資源は，キーワード nonrenewable を用いて，使用した再生不能資源の量の上限を定めます．

```
nonrenewable
  消費量（作業名, 処理モード名）
  消費量（作業名, 処理モード名）
   ...   <= 供給量
```

記述のない作業・処理モードの組に対しては, 消費量 0 と見なされます．なお，消費量は負の値であってもよいものとします．負の場合には供給量とみなされます．

例題において，再生不能資源を 4 万円以下にしたい場合には，以下のように書きます．

```
nonrenewable   weight inf
          +1 (activity[1],m[1][2]) +1 (activity[2],m[2][2])
          +1 (activity[3],m[3][2]) +1 (activity[4],m[4][2])
          +1 (activity[5],m[5][2])            <= 4
```

ここで weight inf は制約を逸脱したときのペナルティ(重み）を無限大と設定していることを表します．すなわち，この制約の逸脱は絶対に許さないことを指定し，逸脱をしない解がない場合には，実行不能 (infeasible) という結果を返します．制約の逸脱を許す場合には，重みを正数値として入力します．

再生不能資源の上限が 4 のときのスケジュールは，図 16 上図のようになります．完了時刻（離陸時刻）は 45 分後で，作業 5 以外の作業はすべて短縮モードで行われます．よって，かかったお金は 4 万円となります．

次に，再生不能資源の上限を 1 にすると，図 16 中図にようなスケジュールが得られます．完了時刻（離陸時刻）は 52 分後で，作業 3 のみが短縮モードで行われます．かかったお金は 1 万円となります．

再生不能資源の上限を 0 にすると，図 16 下図にようなスケジュールが得られます．完了時刻（離陸時刻）は 55 分後で，すべての作業が通常モードで行われます．かかったお金は，当然 0 万円となります．

念のため入力全体を以下に示します．

```
# 航空機をもっと早く離陸させよう!（CPM）

mode m[1][1] duration 13
mode m[1][2] duration 10

mode m[2][1] duration 25
mode m[2][2] duration 20

mode m[3][1] duration 15
mode m[3][2] duration 10

mode m[4][1] duration 27
mode m[4][2] duration 25
```

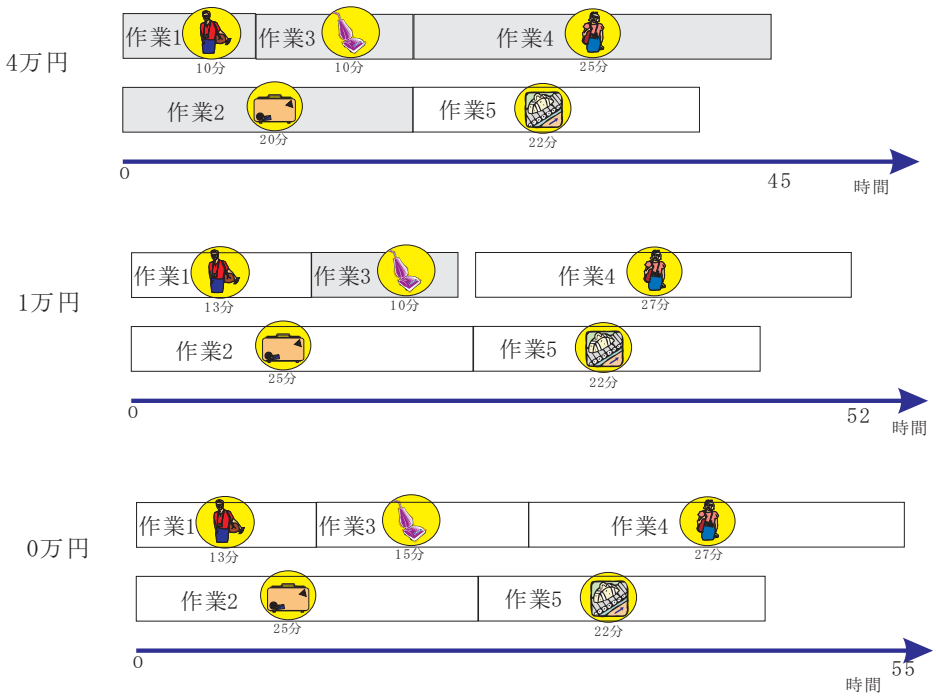

図 16: 再生不能資源の上限が $4, 1, 0$ のときのスケジュール（色のついた矩形で表された作業は，短縮モードで実施されている.）

```
mode m[5][1] duration 22
mode m[5][2] duration 20

activity activity[1]
        m[1][1] m[1][2]

activity activity[2]
        m[2][1] m[2][2]

activity activity[3]
        m[3][1] m[3][2]

activity activity[4]
        m[4][1] m[4][2]

activity activity[5]
        m[5][1] m[5][2]

# 先行制約
temporal  activity[1] activity[3]
temporal  activity[2] activity[4]
temporal  activity[2] activity[5]
temporal  activity[3] activity[4]

# 最大完了時刻最小化
activity sink duedate 0

nonrenewable weight inf
          +1 (activity[1],m[1][2]) +1 (activity[2],m[2][2])
          +1 (activity[3],m[3][2]) +1 (activity[4],m[4][2])
```

```
          +1 (activity[5],m[5][2])            <= 4
```

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう．

再生不能資源の追加もモデルの addResource メソッドを用いて追加します．まず，("資源名",rhs=右辺,direction="制約の向き")（ 1 行目）を引数として入力します．次に，addTerms(資源使用量, 作業, モード)（ 7 行目）を用いて左辺に関する入力をします．作業とモードの追加方法は，前節までの例題と同じです．

```
1 res=m1.addResource("money",rhs=4,direction="<=",weight="inf")
2 for i in durationA:
3     act[i]=m1.addActivity("Act[%s]"%i)
4     mode[i,1]=Mode("Mode[%s][1]"%i,durationA[i])
5     mode[i,2]=Mode("Mode[%s][2]"%i,durationB[i])
6     act[i].addModes(mode[i,1],mode[i,2])
7     res.addTerms(1,act[i],mode[i,2])
```

以下にすべての Python プログラムをまとめておきます．

```
from optseq2 import *

m1=Model()
durationA = {1:13, 2:25, 3:15, 4:27, 5:22 }
durationB = {1:10, 2:20, 3:10, 4:25, 5:20 }

act={}
mode={}
res=m1.addResource("money",rhs=4,direction="<=")
for i in durationA:
    act[i]=m1.addActivity("Act[%s]"%i)
    mode[i,1]=Mode("Mode[%s][1]"%i,durationA[i])
    mode[i,2]=Mode("Mode[%s][2]"%i,durationB[i])
    act[i].addModes(mode[i,1],mode[i,2])
    res.addTerms(1,act[i],mode[i,2])

#temporal (precedense) constraints
m1.addTemporal(act[1],act[3])
m1.addTemporal(act[2],act[4])
m1.addTemporal(act[2],act[5])
m1.addTemporal(act[3],act[4])

print m1
m1.Params.TimeLimit=1
m1.Params.OutputFlag=True
m1.Params.Makespan=True
m1.optimize()
```

練習問題 7 上の例題で，作業時間と短縮したときの費用が，以下のように設定されている場合をモデル化してみましょう．

作業 1： 乗客降ろし 13 分．12 分に短縮可能で，1 万円必要．11 分に短縮するには，さらに 1 万円必要．

作業 2： 荷物降ろし 25 分．23 分に短縮可能で，1 万円必要．21 分に短縮するには，さらに 1 万円必要．

作業 3： 機内清掃 15 分．13 分に短縮可能で， 1 万円必要．11 分に短縮するには，さらに 1 万円必要．

作業 4： 新しい乗客の搭乗 27 分．26 分に短縮可能で， 1 万円必要．25 分に短縮するには，さらに 1 万円必要．

作業 5： 新しい荷物積み込み 22 分．21 分に短縮可能で，1 万円必要．20 分に短縮するには，さらに 1 万円必要．

10　時間制約 –先行制約の一般化–

ここで学ぶこと

- 時間制約の概念と使用法（制約タイプと時間ずれ）
- 時間制約の応用

OptSeq II では，通常の先行制約の他に，様々な時間に関する制約を準備しています．時間制約を用いることによって，実際問題で発生する様々な付加条件をモデル化することができます．

時間制約は，通常の先行制約と同様に，キーワード temporal を用いて，以下のように記述します．

`temporal`　先行作業　後続作業　`type` 制約タイプ　`delay` 時間ずれ

ここで，新しいキーワードである type と delay が追加されました．type の後には，時間制約のタイプ（制約タイプ）を入力し，delay の後には，時間の差（時間ずれ）を入力します．

制約タイプ（type の後の文字）は，先行（後続）作業がの**開始時刻**（start time）を対象にするのか，**完了時刻**（completion time）を対象にするのかを定めます． 制約タイプは `SS`, `SC`, `CS`, `CC` のいずれかを指定します．“S”は開始時刻，“C”は完了時刻を表し，最初に書かれた “S” もしくは “C” が先行作業の対象時刻，後に書かれた “S” もしくは “C” が後続作業の対象時刻を表します．

時間ずれ（delay の後の数字）は，先行作業の時刻と後続作業の時刻の差を指定します．つまり，時間制約は，

(先行作業の開始もしくは完了時刻)+ (時間ずれ) ≤ (後続作業の開始もしくは完了時刻)

を表します．なお，時間ずれは負の値を設定することも可能です．

たとえば `type SS` は, (開始時刻) + (時間ずれ) ≤ (開始時刻) の形の制約を意味します．なお, `type` および `delay` の記述は省略可能であり, その場合には，通常の先行制約（`type CS`, `delay 0`）と見なされます．

時間制約は先行制約と同様の矢印の上に時間ずれと制約タイプを記述します（図 17）.「時間ずれ（制約タイプ）」の書式が，矢印の上にある場合には，その記述にしたがい，省略された場合には，デフォルトの制約タイプ（CS ; 終了-開始）と時間ずれ（0）とします．

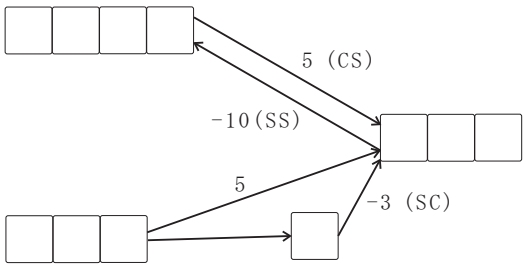

図 17: 時間制約の例

時間制約の適用例として，§ 3 の PERT の例題において，作業 3 と作業 5 の開始時刻が一致しなければならないという制約を考えてみましょう．

開始時刻を一致させるためには，制約タイプは SS（開始-開始の関係）とし，時間ずれは 0 と設定します．また，制約は「作業 3 の開始時刻 ≤ 作業 5 の開始時刻」と「作業 5 の開始時刻 ≤ 作業 3 の開始時刻」の 2 本追加します．

```
temporal  activity[3] activity[5] type SS delay 0
temporal  activity[5] activity[3] type SS delay 0
```

このような制約を付加して求解すると，以下のような結果が得られます．

```
--- best activity list ---
source activity[2] activity[1] activity[3] activity[5] activity[4] sink

--- best solution ---
source ---: 0 0
sink ---: 67 67
activity[1] ---: 0 0--13 13
activity[2] ---: 0 0--25 25
activity[3] ---: 25 25--40 40
activity[4] ---: 40 40--67 67
activity[5] ---: 25 25--47 47

objective value = 67
cpu time = 0.00/3.00(s)
iteration = 0/85383
```

確かに，作業 3 と作業 5 の作業開始時刻が一致していることがわかります．図 18 にガントチャートを示します．

図 18: PERT の例題に作業 3 と作業 5 の同時開始を付加した場合の解（ガントチャート）

また，作業の開始時間の固定も，この時間制約を用いると簡単にできます．OptSeq II では，すべての作業に先行するダミーの作業 source を準備してあります．この作業は必ず時刻 0 に処理されるので，開始時刻に相当する時間ずれをもつ時間制約を 2 本追加することによって，開始時刻を固定できます．

たとえば，作業 5（名称は activity[5]）の開始時刻を 50 分に固定したい場合には，

```
temporal  source activity[5] type SS delay 50
temporal  activity[5] source type SS delay -50
```

と時間制約を追加します．これは，作業 5 の開始時刻が source の開始時刻（0）の 50 分後以降であることと，source の開始時刻が作業 5 の開始時刻の −50 分後以前（言い換えれば，作業 5 の開始時刻が 50 分後以降）であることを規定することを意味します．

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう．

時間制約の追加は先行制約と同様にモデルの `addTemporal` メソッドを用いて追加します．引数として，(先行作業，後続作業,"制約タイプ", 時間ずれ) を入力します．例題のように作業 3 と作業 5 の開始時刻を一致させるには，以下のような 2 本の制約が必要です．

```
m1.addTemporal(act[3],act[5],"SS",0)
m1.addTemporal(act[5],act[3],"SS",0)
```

すべての作業に先行する（時刻 0 に処理される）ダミーの作業 source に対する時間制約を付加したいときには，文字列の`"source"`を先行（後続）作業オブジェクトのかわりに引数に入力します．たとえば，作業 5（オブジェクトは act[5]）の開始時刻を 50 分に固定したい場合には，以下のように 2 つの時間制約を定義します．

```
m1.addTemporal("source",act[5],"SS",delay=50)
m1.addTemporal(act[5],"source","SS",delay=-50)
```

以下にすべての Python プログラムをまとめておきます．

```
from optseq2 import *

m1=Model()
durationA = {1:13, 2:25, 3:15, 4:27, 5:22 }

act={}
mode={}
for i in durationA:
    act[i]=m1.addActivity("Act[%s]"%i)
    mode[i]=Mode("Mode[%s]"%i,durationA[i])
    act[i].addModes(mode[i])

#temporal (precedense) constraints
m1.addTemporal(act[1],act[3])
m1.addTemporal(act[2],act[4])
m1.addTemporal(act[2],act[5])
m1.addTemporal(act[3],act[4])

#act[3] and act[5] start at the same time
m1.addTemporal(act[3],act[5],"SS",0)
m1.addTemporal(act[5],act[3],"SS",0)

print m1
m1.Params.TimeLimit=1
m1.Params.OutputFlag=True
m1.Params.Makespan=True
m1.optimize()
```

練習問題 8 作業 A と作業 B が同時に終了しなければならないことを，OptSeq II で表現してみましょう．

練習問題 9 作業 A が作業を開始した MAX 時間以内に，作業 B の作業を開始しなければならないことを，OptSeq II で表現してみましょう．

練習問題 10 作業を開始する時刻がある時刻より後である必要がある時刻を「リリース時刻」とよびます．§ 7 のお家を作る例題で，内装作業を行うのが 5 日以降でなければならないときのスケジュールを求めてみましょう．

11　作業の途中中断

ここで学ぶこと

- 作業の途中中断の概念と使用法
- 中断中の資源使用量の記述法

多くの実際問題では，緊急の作業などが入ってくると，いま行っている作業を途中で中断して，別の（緊急で行わなければならない）作業を行った後に，再び中断していた作業を途中から行うことがしばしばあります．このように，途中で作業を中断しても，再び（一から作業をやり直すのではなく）途中から作業を続行すること「作業の途中中断」とよびます．

OptSeq II では，文字どおり作業を分割して処理します．たとえば，作業時間が 3 時間の作業があったとします．時間の基本単位を 1 時間としたとき，この作業は，1 時間の作業時間をもつ 3 つの小作業に分割されます．

ただし，作業の一部分が，途中で中断することが難しい場合もよくあります．たとえば，料理をするときに，材料を切ったり，混ぜたりするときには，途中で中断することも可能ですが，いったんオーブンレンジに入れたら，途中でとめたりすることはできません．

OptSeq II では，作業（モード）の時間の区間に対して，最大中断可能時間を入力することによって，様々な作業の**中断**（break）を表現します．これは，作業もしくはモードの定義の中で，以下のようにキーワード break を用いて記述されます．

```
break interval 区間 max 最大中断時間
       interval 区間 max 最大中断時間
       ...
```

interval の後には,（作業開始を 0 と数えたときの）中断可能な開始時刻と終了時刻の組（区間）を入力します．max の後には，中断可能な最大の時間を入力します．この最大中断時間に 0 を入れると，その時間帯における作業の中断ができないことを表します．最大中断可能時間の指定は省略可能であり, その場合 `max inf`（すなわちどれだけ中断しもかまわない）と見なされます．

途中中断の可否は小作業の間に $\bigwedge$ を入れて表します（図 17）．記号 $\bigwedge$ の下の数字は，中断可能な最大時間を表し，省略された場合には，無限大（∞）の中断可能を表します．

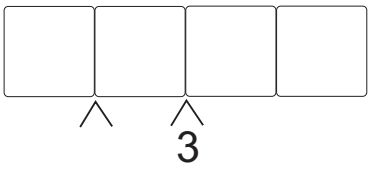

図 19: 作業の途中中断の例

たとえば，§ 8 の納期遅れ最小化問題において，作業 B と作業 C がいつでも最大 1 日だけ中断できることを表すには，

```
activity C duedate 6
        mode  duration 3
        writer interval 0 3 requirement 1
        break interval 0 3 max 1
```

と記述します.

4 日目と 7 日目と 11 日目に休暇を入れたときの納期遅れ最小化問題を解くための入力データは，以下のように書くことができます.

```
# 納期遅れを最小にしよう!（中断あり）

resource writer
  interval 0 3 capacity 1
  interval 4  6 capacity 1
  interval 7 10 capacity 1
  interval 11 inf capacity 1

activity A duedate 5
        mode  duration 1
        writer interval 0 1 requirement 1

activity B duedate 9
        mode  duration 2
        writer interval 0 2 requirement 1
        break interval 0 2 max 1

activity C duedate 6
        mode  duration 3
        writer interval 0 3 requirement 1
        break interval 0 3 max 1

activity D duedate 4
        mode  duration 4
```

```
        writer interval 0 4 requirement 1
        break interval 0 4 max 1
```

上のデータを OptSeq II によって解くと，以下の結果が得られます．

```
--- best activity list ---
source D A B C sink

--- best solution ---
source ---: 0 0
sink ---: 13 13
A ---: 5 5--6 6
B ---: 6 7--9 9
C ---: 8 9--10 11--13 13
D ---: 0 0--3 4--5 5

objective value = 9
cpu time = 0.00/3.00(s)
iteration = 1/38423
```

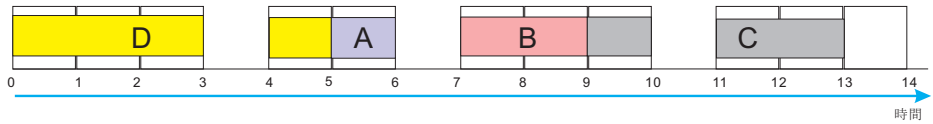

図 20: 途中中断を許した場合の作家の作業順

作業 C と作業 D が休日を挟んで分割して処理されていることが分かります（図 20）．

また，段取りを伴う生産現場においては，中断の途中で他の作業を行うことが禁止されている場合があります．これは，休日の間に異なる作業を行うと，再び段取りなどの処理を行う必要があるため，作業を一からやり直さなければならないからです．

これは，作業の中断中でも資源を使い続けていると表現することによって回避することができます．

中断中の資源の使用量を定義するには，作業（もしくはモード）の記述の中で，

資源名 `interval break` 区間 `requirement` 使用量

`interval break` 区間 `requirement` 使用量

`...`

と入力します．

たとえば，上の作業 C が中断中も資源を 1 単位使用すると定義するには，以下のように書きます．

```
activity C duedate 6
        mode  duration 3
        writer interval 0 3 requirement 1
        writer interval break 0 3 requirement 1
        break interval 0 3 max 1
```

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう.

作業の途中中断は，addBreak(区間の開始時刻, 区間の終了時刻, 最大中断時間) を用いて追加します（5 行目）. また，中断中に資源を使用する場合も通常の資源を追加するのと同様に addResource メソッドを用いて追加します. この場合，引数として (資源,(区間):資源使用量,"break") を入力します. ここで"break"は，前述の資源が中断中に使われている資源であることを表します. たとえば，すべての作業が中断中も資源を 1 単位使用することを表すには，mode[i].addResource(res,(0,"inf"):1,"break")（6 行目）のように入力します.

```
1 for i in duration:
2     act[i]=m1.addActivity("Act[%s]"%i,duedate=due[i])
3     mode[i]=Mode("Mode[%s]"%i,duration[i])
4     mode[i].addResource(res,{(0,"inf"):1})
5     mode[i].addBreak(0,"inf",1)
6     # mode[i].addResource(res,{(0,"inf"):1},"break")
7     act[i].addModes(mode[i])
```

以下にすべての Python プログラムをまとめておきます.

```
from optseq2 import *

m1=Model()

due={1:5,2:9,3:6,4:4}
duration={1:1, 2:2, 3:3, 4:4 }

res=m1.addResource("writer")
res.addCapacity(0,3,1)
res.addCapacity(4,6,1)
res.addCapacity(7,10,1)
res.addCapacity(11,"inf",1)

act={}
mode={}

for i in duration:
    act[i]=m1.addActivity("Act[%s]"%i,duedate=due[i])
    mode[i]=Mode("Mode[%s]"%i,duration[i])
    mode[i].addResource(res,{(0,"inf"):1})
    mode[i].addBreak(0,"inf",1)
    # mode[i].addResource(res,{(0,"inf"):1},"break")
    act[i].addModes(mode[i])

m1.Params.TimeLimit=1
m1.Params.OutputFlag=True
m1.Params.Makespan=False
m1.optimize()
m1.write("chart9.txt")
```

上のプログラムでも最適化を行った後に，m1.write("ファイル名") を入力することによって，簡易ガントチャートを書き出しています. 簡易ガントチャートの出力結果は，以下のようになります.

12 作業の並列処理

ここで学ぶこと

- 作業の並列処理の概念と使用法（最大並列数）
- 並列処理中の資源量（総和と最大資源使用量について）

```
  activity    mode   duration | 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|
-------------------------------------------------------------------------|
  Act[1]   Mode[1]      1     |  |  |  |  |..|==|  |  |  |  |  |  |  |
  Act[2]   Mode[2]      2     |  |  |  |  |  |  |..|==|==|  |  |  |  |
  Act[3]   Mode[3]      3     |  |  |  |  |  |  |  |  |..|==|..|==|==|
  Act[4]   Mode[4]      4     |==|==|==|..|==|  |  |  |  |  |  |  |  |
-------------------------------------------------------------------------
   resource usage/capacity    |
-------------------------------------------------------------------------
            writer            | 1| 1| 1| 0| 1| 1| 0| 1| 1| 1| 0| 1| 1|
                              | 1| 1| 1| 0| 1| 1| 0| 1| 1| 1| 0| 1| 1|
-------------------------------------------------------------------------
```

図 21: 例題の簡易ガントチャート表示．==は活動を処理中を，. . は中断中を表します．

§ 6 で解説した並列ショップスケジューリング問題では，複数の機械（作業員）によって作業時間が短縮されることを，複数のモードを用いることによって表現していました．ここでは，複数資源による作業の並列処理を，より簡単に表現するための方法を紹介します．

前節の作業の途中中断と同じように，作業を，単位時間の作業時間をもつ小作業に分解して考えます．いま，資源使用量の上限が 1 より大きいとき，分解された小作業は，並列して処理できるものとします．ただし，無制限に並列処理ができない場合も多々あるので，OptSeq II では，最大並列数とよばれるパラメータを用いて表現します．

並列処理は，作業（もしくはモード）の記述の中で，以下のように並列が可能な区間（小作業の番号；1 から開始される番号であることに注意）と最大並列数を設定することによって記述します．

```
parallel interval 開始小作業番号  終了小作業番号  max 最大並列数
         interval 開始小作業番号  終了小作業番号  max 最大並列数
         ...
```

最大並列処理可能数の指定は省略可能であり, その場合 `max inf`（資源の上限を超えないならいくらでも並列処理可能）と見なされます. 例えば

```
parallel interval 1 1 max 3
         interval 2 3 max 2
```

は, 最初の小作業は最大 3 個，2 番目, 3 番目の小作業から最大 2 個の小作業を並列処理可能であることを意味します. 並列処理は小作業を表す矩形の上に，点線の矩形を入れて表します（図 22）．点線の矩形の数だけ，通常の作業の上に重ねて処理できることを表すので，図 22 の例では，図右に示してある作業の組み合わせの中で，最も良いものが選択されます．

複数の小作業が並列処理される場合, デフォルトでは, その間の資源使用量は各小作業が使用する量の総和と見なされますが,

```
資源名  max interval 区間  requirement  使用量
        max interval 区間  requirement  使用量
```

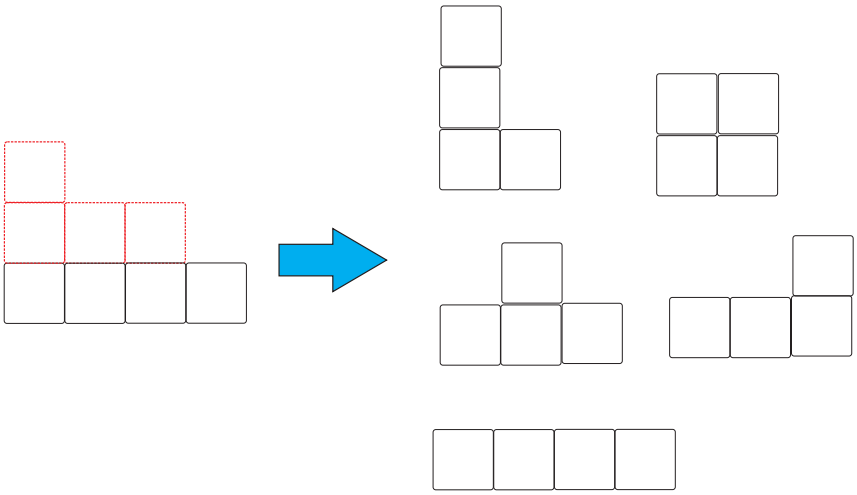

図 22: 並列作業の設定例

...

とキーワード max を付加して記述することで, 資源使用量を総和ではなく, 最大量とすることもできます.

§ 6 の並列ショップスケジューリング問題において，給油作業（作業時間 3 秒）を，最初の（1 番目の）小作業から最大 3 個並列可能とした場合の，入力データは以下のようになります.

```
# 並列ショップスケジューリングの例題（作業の並列処理の例）

resource worker  interval 0 inf  capacity 3

activity prepare
        mode duration 3
        parallel interval 1 1 max 3        #これが追加された.
        worker interval 0 3 requirement 1

activity water
        mode  duration 2
        worker interval 0 2 requirement 1

activity front
        mode  duration 2
        worker interval 0 2 requirement 1

activity jackup
        mode  duration 2
        worker interval 0 2 requirement 1

activity tire1
        mode  duration 4
        worker interval 0 4 requirement 1

activity tire2
        mode  duration 4
        worker interval 0 4 requirement 1

activity tire3
        mode  duration 4
```

```
        worker interval 0 4 requirement 1

activity tire4
        mode  duration 4
        worker interval 0 4 requirement 1

activity oil
        mode  duration 11
        worker interval 0 11 requirement 1

activity jackdown
        mode  duration 2
        worker interval 0 2 requirement 1

# 先行制約
temporal  prepare oil
temporal  jackup tire1
temporal  jackup tire2
temporal  jackup tire3
temporal  jackup tire4
temporal  tire1 jackdown
temporal  tire2 jackdown
temporal  tire3 jackdown
temporal  tire4 jackdown

# 最大完了時刻最小化
activity sink duedate 0
```

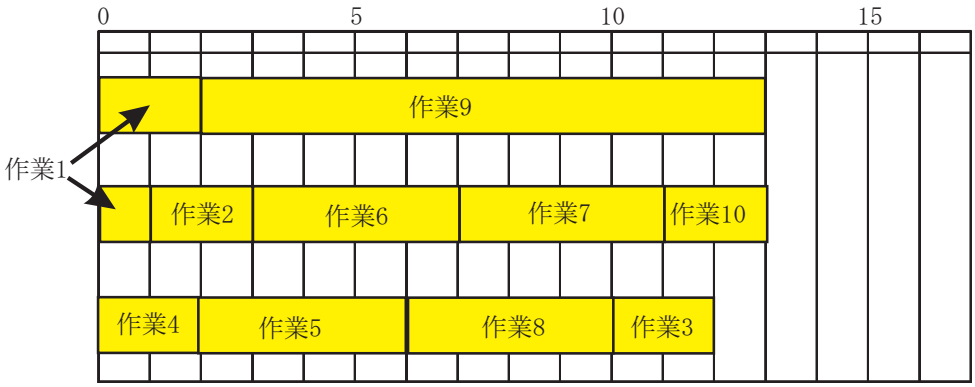

図 23: 作業 1 の並列作業を許した場合のスケジューリング

計算結果は以下のようになり，1 秒短縮して 13 秒で作業が終了することが確認できます（図 23）.

```
--- best activity list ---
source jackup tire1 prepare water oil tire2 tire4 tire3 front jackdown sink

--- best solution ---
source ---: 0 0
sink ---: 13 13
prepare ---: 0 0--1[2] 1--2 2
water ---: 1 1--3 3
front ---: 10 10--12 12
jackup ---: 0 0--2 2
tire1 ---: 2 2--6 6
tire2 ---: 3 3--7 7
```

```
tire3 ---: 7 7--11 11
tire4 ---: 6 6--10 10
oil ---: 2 2--13 13
jackdown ---: 11 11--13 13

objective value = 13
cpu time = 0.00/3.00(s)
iteration = 25/23174
```

ここで，並列で行われた作業（名称は prepare）においては，並列数が [ ] で表示されます.「0 0–1[2] 1–2 2」は、「時刻 0 に処理を開始し，時刻 0〜1 は並列数 2 で処理し，その後，時刻 1〜2 に並列なしで処理し，時刻 2 で完了」を表します.

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう.

作業の並列処理は，`addParallel(開始小作業番号, 終了小作業番号, 最大並列数)` を用いて追加します（ 6 行目）.

```
1 for i in duration:
2     act[i]=m1.addActivity("Act[%s]"%i)
3     mode[i]=Mode("Mode[%s]"%i,duration[i])
4     mode[i].addResource(res,{(0,"inf"):1})
5     if i==1:
6         mode[i].addParallel(1,1,3)
7     act[i].addModes(mode[i])
```

以下にすべての Python プログラムをまとめておきます.

```
from optseq2 import *

m1=Model()
duration ={1:3, 2:2, 3:2, 4:2, 5:4, 6:4, 7:4, 8:4, 9:11, 10:2 }
res=m1.addResource("worker")
res.addCapacity(0,"inf",3)

act={}
mode={}
for i in duration:
    act[i]=m1.addActivity("Act[%s]"%i)
    mode[i]=Mode("Mode[%s]"%i,duration[i])
    mode[i].addResource(res,{(0,"inf"):1})
    if i==1:
        mode[i].addParallel(1,1,3)
    act[i].addModes(mode[i])

#temporal (precedense) constraints
m1.addTemporal(act[1],act[9])
for i in range(5,9):
    m1.addTemporal(act[4],act[i])
    m1.addTemporal(act[i],act[10])

print m1
m1.Params.TimeLimit=1
m1.Params.OutputFlag=True
m1.Params.Makespan=True
m1.optimize()
m1.write("chart10.txt")
```

上のプログラムでも最適化を行った後に，`m1.write("ファイル名")` を入力することによって，簡易ガントチャートを書き出しています．簡易ガントチャートの出力結果は，以下のようになります.

```
  activity    mode   duration | 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|
-------------------------------------------------------------------------|
 Act[10]   Mode[10]     2     |  |  |  |  |  |  |  |  |  |  |==|==|
  Act[1]   Mode[1]      3     |*2|==|  |  |  |  |  |  |  |  |  |  |  |
  Act[2]   Mode[2]      2     |  |==|==|  |  |  |  |  |  |  |  |  |  |
  Act[3]   Mode[3]      2     |  |  |  |  |  |  |  |  |  |  |==|==|  |
  Act[4]   Mode[4]      2     |==|==|  |  |  |  |  |  |  |  |  |  |  |
  Act[5]   Mode[5]      4     |  |  |==|==|==|==|  |  |  |  |  |  |  |
  Act[6]   Mode[6]      4     |  |  |  |==|==|==|==|  |  |  |  |  |  |
  Act[7]   Mode[7]      4     |  |  |  |  |  |  |  |==|==|==|==|  |  |
  Act[8]   Mode[8]      4     |  |  |  |  |  |  |==|==|==|==|  |  |  |
  Act[9]   Mode[9]      11    |  |  |==|==|==|==|==|==|==|==|==|==|==|
-------------------------------------------------------------------------
   resource usage/capacity    |
-------------------------------------------------------------------------
            worker            | 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 2|
                              | 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3| 3|
-------------------------------------------------------------------------
```

図 24: 例題の簡易ガントチャート表示．==は活動を処理中を，*2 は 2 台の機械で並列実行中を表します．

13 状態変数を使ってみよう！

OptSeq II では，より高度なモデル化をするための機能が搭載されています．ここでは，その中の状態変数の使い方について学習することにします．

ここで学ぶこと

- 状態変数とは
- 状態変数を用いた順序依存の段取り時間の表現法

状態変数とは，時間の経過とともに値（＝状態，非負整数）が変化する変数です．状態を使うことによって，通常のスケジューリングモデルでは表現しきれない，様々な付加条件をモデル化することができます．

状態は，基本的には 1 つ前の時刻の値を引き継ぎますが，あらかじめ指定された時刻に変化させることができます．

具体的な記述方法は以下の通りです．

```
state 状態名
    time 時刻  value 値
    time 時刻  value 値
    time 時刻  value 値
    ...
```

ここで,「time 時刻 value 値」の部分はオプションで，指定時刻 time に指定された値 value（非負整数）に設定することを表します．オプションの「time 時刻 value 値」を省略した場合には，状態の名称だけが与えられたことになり，時刻 0 で状態 0 の状態が定義されます．

状態は，ある作業がある処理モードで開始された直後に変化させることができます．具体的には，作業の処理モードの記述の中で，

状態名 from 値 1 to 値 2

を追加することで，開始時の状態が「値 1」でなくてはならず，開始直後（開始時刻が t であれば $t+1$ に）状態が「値 2」に変化することになります．これによって，処理モードに「ある状態の時しか開始できない」といった制約を加えることができます．この記述は，from 値を変えて複数回行うことができます．例えば，状態変数 State1 の取り得る値が $1,2,3$ の 3 種類としたとき，

```
State1  from 1 to 2
State1  from 2 to 3
State1  from 3 to 1
```

とすれば，開始時点での状態に関係なく処理は可能で，状態は巡回するように 1 つずつ変化することになります．

また，これを利用して「あるタイプの作業（以下の Act1, Act2, Act3）は 1 週間に高々2 回しか処理できない」ことは，以下のように表すことができます．

```
state AtMost2
    time 0   value 0
    time 7   value 0
    time 14  value 0
    ...
activity Act1
    mode  duration 1
      AtMost2  from 0 to 1
      AtMost2  from 1 to 2
      AtMost2  from 2 to 3
activity Act2
    mode  duration 1
      AtMost2  from 0 to 1
      AtMost2  from 1 to 2
      AtMost2  from 2 to 3
activity Act3
    mode  duration 1
      AtMost2  from 0 to 1
      AtMost2  from 1 to 2
```

```
    AtMost2  from 2 to 3
```

なお，作業の定義において，上述のような「開始時状態が指定された処理モード」を与える場合，通常，以下のように「autoselect」を記述することが効率的と思われます.

```
activity 作業名
        autoselect
        処理モード名 処理モード名 ...
```

OptSeq II では,「まず各作業の処理モードをそれぞれ選び，その後，ある順序にしたがって作業を前づめにスケジュールしていく」ということ繰り返し行ないます．したがって,「開始時状態が指定された処理モード」が存在すると，処理モードの選び方によっては,「スケジュールを生成していくとき，割り付け不可能」ということが頻繁に起こり得ます.

これを防ぐため,「あらかじめ処理モードを指定せず，前づめスケジュールしながら適切な処理モードを選択する」ことが必要になり，これを実現するのが「autoselect」です.

注意！作業の定義に autoselect を指定した場合には，その作業に制約を逸脱したときの重みを無限大とした（すなわち絶対制約とした）再生不能資源を定義することはできません．かならず重みを無限大ではない正数値と設定して下さい.

この機能の利用例として，順序依存の段取り作業があります.

例の概要は以下のとおりです.

- 資源は機械 1 台
- 作業は A,B の 2 つのタイプがあり，それぞれ 3 個ずつの計 6 個．処理時間はすべて 3
- タイプの異なる作業間の段取り時間は 5，同タイプであれば段取り時間 1
- メイクスパン（最大完了時刻）最小化が目的

これを実現するために,

- 段取り用状態変数を定義．値 1 は A タイプ作業の処理直後に，値 2 は B タイプ作業の処理直後に対応
- 各作業に対して，段取り作業を定義
- A タイプ作業の段取り作業には，AtoA，BtoA の 2 つのモードを設定
- B タイプ作業の段取り作業には，AtoB，BtoB の 2 つのモードを設定
- 段取り作業と本作業の間に他の作業が割り込まないように,「段取り作業の完了時刻 = 本作業の開始時刻」となるよう時間制約を追加し，さらに，本作業は開始直後に中断可能（中断中も資源消費）となるよう記述

としています.

```
# 資源は機械のみ
resource machine
 interval 0 inf capacity 1

# 段取り用状態変数
#   value = 1（A タイプ作業後） or 2（B タイプ作業後）
#   初期状態は 1
state SetupState
      time 0 value 1

# 段取り作業用処理モード（4 タイプ： A-->A, A-->B, B-->A, B-->B）
mode SetupAtoA duration 1
     machine  interval 0 1 requirement 1
     SetupState from 1 to 1

mode SetupAtoB duration 5
     machine  interval 0 5 requirement 1
     SetupState from 1 to 2

mode SetupBtoB duration 1
     machine  interval 0 1 requirement 1
     SetupState from 2 to 2

mode SetupBtoA duration 5
     machine  interval 0 5 requirement 1
     SetupState from 2 to 1


# 段取り作業＋本作業（A タイプ, B タイプそれぞれ 3 組）
activity setupA[1]
 autoselect
 SetupAtoA SetupBtoA
activity A[1]
 mode duration 3
 break interval 0 0
 machine interval 0 3 requirement 1
 machine interval break 0 0 requirement 1

activity setupA[2]
 autoselect
 SetupAtoA SetupBtoA
activity A[2]
 mode duration 3
 break interval 0 0
 machine interval 0 3 requirement 1
 machine interval break 0 0 requirement 1

activity setupA[3]
 autoselect
 SetupAtoA SetupBtoA
activity A[3]
 mode duration 3
 break interval 0 0
 machine interval 0 3 requirement 1
 machine interval break 0 0 requirement 1


activity setupB[1]
 autoselect
```

```
 SetupAtoB SetupBtoB
activity B[1]
 mode duration 3
 break interval 0 0
 machine interval 0 3 requirement 1
 machine interval break 0 0 requirement 1

activity setupB[2]
 autoselect
 SetupAtoB SetupBtoB
activity B[2]
 mode duration 3
 break interval 0 0
 machine interval 0 3 requirement 1
 machine interval break 0 0 requirement 1

activity setupB[3]
 autoselect
 SetupAtoB SetupBtoB
activity B[3]
 mode duration 3
 break interval 0 0
 machine interval 0 3 requirement 1
 machine interval break 0 0 requirement 1

# 段取り作業と本作業の接続
temporal setupA[1] A[1] type CS
temporal A[1] setupA[1] type SC

temporal setupA[2] A[2] type CS
temporal A[2] setupA[2] type SC

temporal setupA[3] A[3] type CS
temporal A[3] setupA[3] type SC

temporal setupB[1] B[1] type CS
temporal B[1] setupB[1] type SC

temporal setupB[2] B[2] type CS
temporal B[2] setupB[2] type SC

temporal setupB[3] B[3] type CS
temporal B[3] setupB[3] type SC


# 最大完了時刻最小化
activity sink duedate 0
```

これを OptSeq II で解いてみると，結果は以下のようになり，きちんと A タイプの作業をした後に，段取り時間 5 を使って B タイプの作業に切り替えていることが分かります.

```
--- best activity list ---
source setupA[3] A[3] setupA[1] A[1] setupA[2] A[2] setupB[3] B[3] setupB[1] B[1] setupB[2] B[2] sink

--- best solution ---
source ---: 0 0
sink ---: 28 28
setupA[1] SetupAtoA: 4 4--5 5
A[1] ---: 5 5--8 8
```

```
setupA[2] SetupAtoA: 8 8--9 9
A[2] ---: 9 9--12 12
setupA[3] SetupAtoA: 0 0--1 1
A[3] ---: 1 1--4 4
setupB[1] SetupBtoB: 20 20--21 21
B[1] ---: 21 21--24 24
setupB[2] SetupBtoB: 24 24--25 25
B[2] ---: 25 25--28 28
setupB[3] SetupAtoB: 12 12--17 17
B[3] ---: 17 17--20 20

objective value = 28
cpu time = 0.01/1.00(s)
iteration = 7/1814
```

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう．

まず，状態を定義するには，モデルクラスの addState メソッドを用います．addState メソッドの引数は，状態の名称を表す文字列であり，返値は状態オブジェクトです．たとえば，"Setup_State"と名付けた状態 s1 をモデルオブジェクト m1 に追加するには，以下のようにします．

```
s1=m1.addState("Setup_State")
```

状態が，ある時間においてある値に変化することを定義するためには，状態オブジェクトに対する addValue メソッドを用います．たとえば，状態オブジェクト s1 が時刻 0 に値 1 になることを定義するには，以下のようにします．

```
s1.addValue(time=0,value=1)
```

状態の値が特定の値でないとモードが開始できず，さらに開始直後に別の値に変化することを表すには，モードクラスの addState メソッドを用います．このメソッドの引数は，状態オブジェクト (state)，開始時の状態（fromValue; 非負整数），開始直後に変化する状態（toValue; 非負変数）です．

たとえば，モード mode1 を開始したときに状態 s1 の値が 1 から 2 に変わることを表すには，以下のようにします．

```
mode1.addState(s1,1,2)
```

また，作業に対してモードを自動選択に設定するには，作業クラスのコンストラクタの引数 autoselect を用いるか，作業オブジェクトのプロパティautoselect を用います．具体的には，以下の何れの書式でも，自動選択に設定できます．

```
act1=m1.addActivity("Act1",autoselect=True)
act1.autoselect=True
```

上の書式を用いると，例題の Python によるプログラムは，以下のように簡潔に書くことができます．

```
from optseq2 import *
m1=Model()
duration={(1,1):3,(1,2):3,(1,3):3,(2,1):3,(2,2):3,(2,3):3}
setup={(1,1):1,(1,2):5,(2,1):5,(2,2):1}
act={}
act_setup={}
mode={}
mode_setup={}
rs=m1.addResource("machine",1)
s1=m1.addState("Setup_State")
s1.addValue(time=0,value=1)
```

```
activity    mode   duration | 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|
-------------------------------------------------------------------------------------------------------------------|
  Act11     Mode11      3   |  |  |  |  |  |  |  |  |==|==|==|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
  Act12     Mode12      3   |  |  |  |  |  |==|==|==|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
  Act13     Mode13      3   |  |==|==|==|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
  Act21     Mode21      3   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |==|==|==|  |  |  |  |
  Act22     Mode22      3   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |==|==|==|  |  |  |  |  |  |  |
  Act23     Mode23      3   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |==|==|==|
 Setup11  Mode_setup11  1   |  |  |  |  |  |  |  |  |==|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 Setup12  Mode_setup11  1   |  |  |  |  |==|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 Setup13  Mode_setup11  1   |==|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 Setup21  Mode_setup22  1   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |==|  |  |  |  |  |  |
 Setup22  Mode_setup12  5   |  |  |  |  |  |  |  |  |  |  |  |  |==|==|==|==|==|  |  |  |  |  |  |  |  |  |  |
 Setup23  Mode_setup22  1   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |==|  |  |
-------------------------------------------------------------------------------------------------------------------
   resource usage/capacity  |
-------------------------------------------------------------------------------------------------------------------
           machine          | 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
                            | 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1| 1|
-------------------------------------------------------------------------------------------------------------------
```

図 25: 例題の簡易ガントチャート表示.

```
for (i,j) in setup:
    mode_setup[i,j]=Mode("Mode_setup"+str(i)+str(j),setup[i,j])
    mode_setup[i,j].addState(s1,i,j)
    mode_setup[i,j].addResource(rs,{(0,"inf"):1})
for (i,j) in duration:
    act_setup[i,j]=m1.addActivity("Setup"+str(i)+str(j),autoselect=True)
    act_setup[i,j].addModes(mode_setup[1,i],mode_setup[2,i])
    act[i,j]=m1.addActivity("Act"+str(i)+str(j))
    mode[i,j]=Mode("Mode"+str(i)+str(j),duration[i,j])
    mode[i,j].addResource(rs,{(0,"inf"):1})
    mode[i,j].addBreak(0,0)
    mode[i,j].addResource(rs,{(0,"inf"):1},"break")
    act[i,j].addModes(mode[i,j])
for (i,j) in duration:
    m1.addTemporal(act_setup[i,j],act[i,j],"CS")
    m1.addTemporal(act[i,j],act_setup[i,j],"SC")
m1.Params.TimeLimit=1
m1.Params.Makespan=True
m1.optimize()
```

結果は以下のように出力されます.

```
source --- 0 0
sink --- 28 28
Setup12 Mode_setup11 4 5
Act12 --- 5 8
Setup13 Mode_setup11 0 1
Act13 --- 1 4
Setup21 Mode_setup22 20 21
Act21 --- 21 24
Setup23 Mode_setup22 24 25
Act23 --- 25 28
Setup22 Mode_setup12 12 17
Act22 --- 17 20
Setup11 Mode_setup11 8 9
Act11 --- 9 12
planning horizon= 28
```

上のプログラムでも最適化を行った後に，`m1.write("ファイル名")` を入力することによって，簡易ガントチャートを書き出しています．簡易ガントチャートの出力結果は，図 25 のようになります.

14　機械スケジューリング–複雑な問題に挑戦しよう！–

ここで学ぶこと

- 機械スケジューリング（ジョブショップ）について
- 複雑な制約の記述法

機械スケジューリングは，OptSeq II で扱う資源制約付きスケジューリングの特殊形です．機械とは，資源量上限が 1 単位の特殊な資源と考えられ，さらに作業を行うときには，その機械の資源を 1 単位使用すると考えます．このとき，作業は機械を「占有する」とよびます.

機械スケジューリング問題は，古くから多くの研究があり，問題のクラス別にベンチマーク問題とよばれる例題集が完備されています．ここでは，複雑な条件をもつ機械スケジューリング問題例に対して，OptSeq II を適用することによって，良好な解が短時間で得られることを示します.

4 仕事 3 機械のジョブショップ機械スケジューリング問題を考えます．各仕事はそれぞれ 3 つの作業 1, 2, 3 から成り, この順序に処理しなくてはならないものとします．各作業を処理する機械, および処理日数は以下の通りです.

	作業 1	作業 2	作業 3
仕事 1	機械 1 / 7 日間	機械 2 / 10 日間	機械 3 / 4 日間
仕事 2	機械 3 / 9 日間	機械 1 / 5 日間	機械 2 / 11 日間
仕事 3	機械 1 / 3 日間	機械 3 / 9 日間	機械 2 / 12 日間
仕事 4	機械 2 / 6 日間	機械 3 / 13 日間	機械 1 / 9 日間

このように，仕事によって作業を行う機械の順番が異なる問題を，**ジョブショップ**（job shop）とよびます（ちなみに，作業順が同じ問題を**フローショップ**（flow shop）とよびます）． 目的は最大完了時間最小化です．ここでは，さらに以下のような複雑な条件がついているものとします.

- 各作業の初め 2 日間は作業員資源を必要とする操作であり, この操作は平日のみ, かつ 1 日あたり高々2 個しか行うことができないものとします.
- 各作業は，1 日経過した後だけ，中断が可能であるとします.
- 機械 2 に限り, 特急処理が可能であるとします．特急処理を行うと処理日数は 4 日で済むが, コストがかかるため, 全体で 1 度しか行うことはできないものと仮定します.
- 機械 1 において, 仕事 1 を処理した後は仕事 2 を処理しなくてはならないものとします.

上の問題例を図示すると図 26 のようになります.

この問題は，機械および作業員資源を再生可能資源とした 12 作業の問題として，OptSeq II で記述できます.

まず，機械資源はずっと使用できるものとし，その供給量（使用可能量の上限）を 1 と設定します.

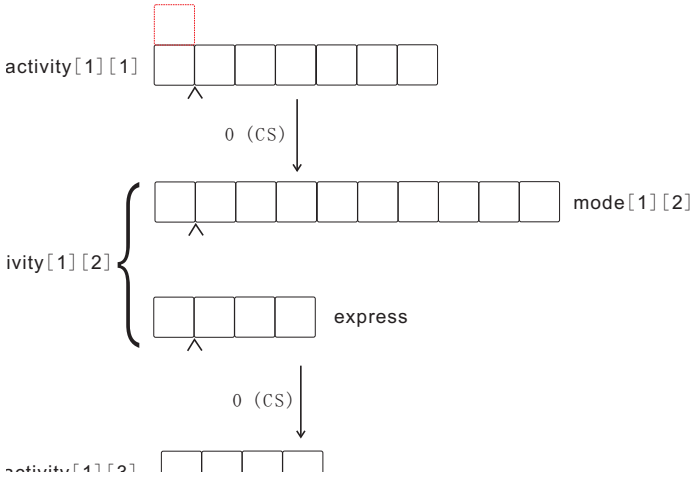

図 26: ジョブショップ問題の例

```
# 機械資源
resource machine[1]  interval 0 inf  capacity 1
resource machine[2]  interval 0 inf  capacity 1
resource machine[3]  interval 0 inf  capacity 1
```

作業員も資源として登録します．この場合には，1 日あたり高々2 個しか行うことができないので，資源の供給量は，平日は 2，休日は 0 名と設定します（最初の日は月曜日と仮定します）．

```
# 人的資源：週末は使用不可
resource manpower
     interval  0   5  capacity 2
     interval  7  12  capacity 2
     interval 14  19  capacity 2
     interval 21  26  capacity 2
     interval 28  33  capacity 2
     interval 35  40  capacity 2
     interval 42  47  capacity 2
     interval 49  54  capacity 2
     interval 56  71  capacity 2
```

次に，作業（activity）の記述を行います．

まず，作業 1 ですが，作業時間は 7 日間，1 日後に途中中断を許し，最初の 1 日だけは並列処理が可能であると定義します．

```
# 作業の定義（activity[2]～activity[4] は省略）
activity activity[1][1]
     # 処理モードが 1 つだけのときは，activity 記述内に mode を記述可能
     mode  duration 7
     break interval 1 1
     parallel interval 1 1 max 2  # machine[1] で並列処理可能
     machine[1] max interval 0 7  requirement 1  # 並列処理中も，必要資源量は 1
     manpower       interval 0 2  requirement 1
```

続いて，作業 2 ですが，これは機械 2 で処理されるので，特急処理を行うことができます．これは，特急処理を行うモード express を準備することによって処理します．特急処理の処理時間は 4 日です．

```
# machine[2] 用特急処理
mode express duration 4
break    interval 1 1   # 1 番目と 2 番目の小作業間で中断可能
        machine[2] max interval 0 4  requirement 1
parallel interval 1 1 max 2
        manpower interval 0 2  requirement 1
```

もう 1 つの通常作業もモード mode[1][2] として定義し，作業 activity[1][2] は，2 つのモードをもつものと定義します．

```
mode mode[1][2] duration 10
break interval 1 1
parallel interval 1 1 max 2
        machine[2] max interval 0 10 requirement 1
        manpower    interval 0 2  requirement 1

activity activity[1][2] mode[1][2]  express
```

作業 3 は，通常の作業だけなので，以下のように簡単に記述できます．

```
activity activity[1][3]
        mode  duration 4
break interval 1 1
parallel interval 1 1 max 2
        machine[3] max interval 0 4  requirement 1
        manpower   interval 0 2  requirement 1
```

先行制約は，同じ仕事に含まれる作業間に設定されます.

```
# 先行制約
temporal  activity[1][1] activity[1][2]
temporal  activity[1][2] activity[1][3]

temporal  activity[2][1] activity[2][2]
temporal  activity[2][2] activity[2][3]

temporal  activity[3][1] activity[3][2]
temporal  activity[3][2] activity[3][3]

temporal  activity[4][1] activity[4][2]
temporal  activity[4][2] activity[4][3]
```

3 番目の条件「機械 1 において, 作業 1 を処理した後は作業 2 を処理しなくてはならない」を記述するためには，多少の工夫が必要です．この制約は，直前先行制約とよばれます．ここでは，以下のようにしてモデル化を行います.

1. 処理時間 0 の仮想作業 dummy を導入し, 時刻 0 で中断可能と設定します. そして, 中断中, 資源「機械 1」を消費し続けるものと定義します.
2. 時間制約を用いて, (作業 1 の完了時刻) = (dummy の開始時刻) および (dummy の完了時刻) = (作業 2 の開始時刻) の 2 つの等式制約を追加します

この結果, 仕事 1・作業 1 の完了後, 仕事 2・作業 2 が開始されるまで機械資源は消費され続けることになり, 他の作業を行うことはできないことになります．OptSeq II のモデルファイルとして記述すると，以下のようになります.

```
# activity[1][1] と activity[2][2] の間を繋ぐ仮想作業
activity act[1][1]_[2][2]
     mode duration 0
     break interval 0 0
     machine[1] interval break 0 0 requirement 1 # 中断中も資源 machine[1] を使用
# activity[1][1] の完了時刻 = act[1][1]_[2][2] の開始時刻
temporal activity[1][1] act[1][1]_[2][2] type CS
temporal act[1][1]_[2][2] activity[1][1] type SC

# act[1][1]_[2][2] の完了時刻 = activity[2][2] の開始時刻
```

```
temporal act[1][1]_[2][2] activity[2][2] type CS
temporal activity[2][2] act[1][1]_[2][2] type SC
```

また，特急処理は高々1 回しかできないことを表すには，再生不能資源制約を用います.

```
# 特急処理は高々1 回
nonrenewable
          +1 (activity[1][2],express) +1 (activity[2][3],express)
          +1 (activity[3][3],express) +1 (activity[4][1],express)
          <= 1
```

最後に，目的である最大完了時刻最小化を記述するために，最後のダミー作業 sink の納期を 0 に設定しておきます.

```
# 最大完了時刻最小化
activity sink duedate 0
```

プログラム終了時に, 探索で得られた最良スケジュール（完了時刻は 33）が表示されます. たとえば, activity[1][1] の開始時刻は 1 で, まず時刻 1〜2 の間に 2 つの小作業が並列処理された後 (1--2[2]), 残りの小作業が時刻 7 まで行われています. また, activity[1][2] の処理モードは mode[1][2] で, 時刻 7〜8 の間に 2 つの小作業が並列処理された後，残りの小作業が時刻 16 まで行われています.

```
# reading data ... done: 0.00(s)
# random seed: 1
# tabu tenure: 1
# cpu time limit: 1.00(s)
# iteration limit: 1073741823
# computing all-pairs longest paths and strongly connected components ... done
#scc 13
objective value = 46 (cpu time = 0.00(s), iteration = 0)
0: 0.00(s): 46/46
objective value = 39 (cpu time = 0.00(s), iteration = 1)
objective value = 38 (cpu time = 0.00(s), iteration = 2)
objective value = 37 (cpu time = 0.00(s), iteration = 4)
objective value = 36 (cpu time = 0.00(s), iteration = 5)
objective value = 34 (cpu time = 0.02(s), iteration = 31)
objective value = 33 (cpu time = 0.02(s), iteration = 33)

--- best activity list ---
source activity[2][1] activity[1][1] act[1][1]_[2][2] activity[2][2] activity[4][1] activity[3][1] activity[1][

--- best solution ---
source ---: 0 0
sink ---: 33 33
activity[1][1] ---: 1 1--2[2] 2--7 7
activity[1][2] mode[1][2]: 7 7--8[2] 8--16 16
activity[1][3] ---: 30 30--31[2] 31--33 33
activity[2][1] ---: 0 0--1[2] 1--8 8
activity[2][2] ---: 8 8--9[2] 9--12 12
activity[2][3] mode[2][3]: 16 16--17[2] 17--26 26
activity[3][1] ---: 14 14--15[2] 15--16 16
```

```
activity[3][2] ---: 21 21--22[2] 22--29 29
activity[3][3] express: 29 29--30[2] 30--32 32
activity[4][1] mode[4][1]: 2 2--3[2] 3--7 7
activity[4][2] ---: 9 9--10[2] 10--21 21
activity[4][3] ---: 22 22--23[2] 23--30 30
act[1][1]_[2][2] ---: 7 8

objective value = 33
cpu time = 0.02/1.00(s)
iteration = 33/1825
```

この例題を Python から OptSeq II を呼び出す方法で解いてみましょう.

まず，モデルオブジェクト `m1` を生成し，機械を表す資源を追加しますこのとき，機械資源の容量（使用可能量の上限）を 1 と設定しておきます.

```
m1=Model()
machine={}
for j in range(1,4):
    machine[j]=m1.addResource("machine[%s]"%j,capacity={(0,"inf"):1})
```

作業員も資源であり，この場合には，1 日あたり高々2 個しか行うことができないので，資源の容量は，平日は 2，休日は 0 名と設定しています（ただし最初の日は月曜日と仮定）.

```
manpower=m1.addResource("manpower")
for t in range(9):
    manpower.addCapacity(t*7,t*7+5,2)
```

最後に，特急処理が高々1 回しか行うことができないことを表すために，予算 `budget` と名付けた再生不能資源を追加し，制約の右辺 `rhs` を 1 に設定しておきます.

```
budget=m1.addResource("budget_constraint",rhs=1)
```

次に，活動とモードに関する記述を行います.

まず，データを保管するために，活動の番号と作業の番号のタプルをキー，機械番号と作業時間のタプルを値とした辞書 `JobInfo` を以下のように準備しておきます.

```
JobInfo={ (1,1):(1,7), (1,2):(2,10), (1,3):(3,4),
          (2,1):(3,9), (2,2):(1,5), (2,3):(2,11),
          (3,1):(1,3), (3,2):(3,9), (3,3):(2,12),
          (4,1):(2,6), (4,2):(3,13), (4,3):(1,9)
    }
```

特急処理を行うモード `express` を準備します（これは機械 2 に限定した処理で作業時間は 4）.

```
1 express=Mode("Express",duration=4)
2 express.addResource(machine[2],{(0,"inf"):1},"max")
3 express.addResource(manpower,{(0,2):1})
4 express.addBreak(1,1)
```

活動とモードは辞書 `act`,`mode` に保管します（1,2 行目）. 6 行目は，並列作業中でも 1 単位の機械資源を使用することを表し，7 行目は，作業員が最初の 2 日間だけ必要なことを表し，8 行目は，1 日経過後に 1 日だけ中断が可能なことを表します. また，機械 1 上では並列処理が可能であり（9,10 行目），機械 2 に対しは通常モードと特急モード `express` を追加し，さらに特急モードで処理した場合には予算資源 `budget` を 1 単位使用するものとします（11 から 13 行目）.

```
1  act={}
2  mode={}
3  for (i,j) in JobInfo:
4      act[i,j]=m1.addActivity("Act[%s][%s]"%(i,j))
5      mode[i,j]=Mode("Mode[%s][%s]"%(i,j),duration=JobInfo[i,j][1])
6      mode[i,j].addResource(machine[JobInfo[i,j][0]],{(0,"inf"):1},"max")
7      mode[i,j].addResource(manpower,{(0,2):1})
8      mode[i,j].addBreak(1,1)
9      if JobInfo[i,j][0]==1:
10         mode[i,j].addParallel(1,1,2)
11     if JobInfo[i,j][0]==2:
```

```
12                 act[i,j].addModes(mode[i,j],express)
13                 budget.addTerms(1,act[i,j],express)
14             else:
15                 act[i,j].addModes(mode[i,j])
```

先行制約（作業の前後関係）を表す時間制約を，同じ活動に含まれる作業間に設定しておきます．

```
for i in range(1,5):
    for j in range(1,3):
        m1.addTemporal(act[i,j],act[i,j+1])
```

条件「機械 1 において, 作業 1 を処理した後は作業 2 を処理しなくてはならない」を記述するためには，多少のモデル化のための工夫が必要です．この制約は，直前先行制約とよばれ，以下のようにしてモデル化を行うことができます．

1. 処理時間 0 のダミーの（仮想の）作業 dummy （以下のモデルファイルでは `d_act`）を導入し, 時刻 0 で中断可能と設定. そして, 中断中, 資源「機械 1」を消費し続けるものと定義.
2. 時間制約を用いて, (作業 1 の完了時刻) = (dummy の開始時刻) および (dummy の完了時刻) = (作業 2 の開始時刻) の 2 つの制約を追加.

この結果, 活動 1・作業 1 `act[1,1]` の完了後, 活動 2・作業 2 `act[2,2]` が開始されるまで機械 1 の資源は消費され続けることになり, 他の作業を行うことはできないことになります.

```
d_act=m1.addActivity("dummy_activity")
d_mode=Mode("dummy_mode")
d_mode.addBreak(0,0)
d_mode.addResource(machine[1],{(0,0):1},"break")
d_act.addModes(d_mode)
m1.addTemporal(act[1,1],d_act,tempType="CS")
m1.addTemporal(d_act,act[1,1],tempType="SC")
m1.addTemporal(d_act,act[2,2],tempType="CS")
m1.addTemporal(act[2,2],d_act,tempType="SC")
```

最後に，目的である最大完了時刻最小化をパラメータ `Makespan` で設定し，計算時間上限 1 秒で求解します後でガントチャートをファイル `chart1.txt` に出力します．

```
m1.Params.TimeLimit=1
m1.Params.Makespan=True
m1.optimize()
m1.write("chart1.txt")
```

実行したときの出力例を以下に示します．プログラム終了時に, 探索で得られた最良スケジュール（完了時刻は 38）が表示されます．たとえば, `Act[2[2]` の開始時刻は 9 で, まず時刻 9〜10 の間に 2 つの小作業が並列処理された後 (`9--10[2]`), 残りの小作業が時刻 13 まで行われます．

```
--- best solution ---
source,---, 0 0
sink,---, 38 38
Act[3][2],---, 23 23--32 32
Act[1][3],---, 32 32--33 35--38 38
Act[2][1],---, 0 0--9 9
Act[2][3],Mode[2][3], 21 21--32 32
Act[4][2],---, 10 10--23 23
Act[1][2],Mode[1][2], 8 8--9 10--19 19
Act[3][3],Express, 32 32--33 35--38 38
Act[3][1],---, 14 14--15[2] 15--16 16
Act[4][3],---, 23 23--32 32
Act[2][2],---, 9 9--10[2] 10--13 13
Act[4][1],Mode[4][1], 2 2--8 8
Act[1][1],---, 0 0--7 7
dummy_activity,---, 7 9
```

最適解を簡易ガントチャートで示したもの (`chart1.txt`) は，次ページの図 27 のようになります．

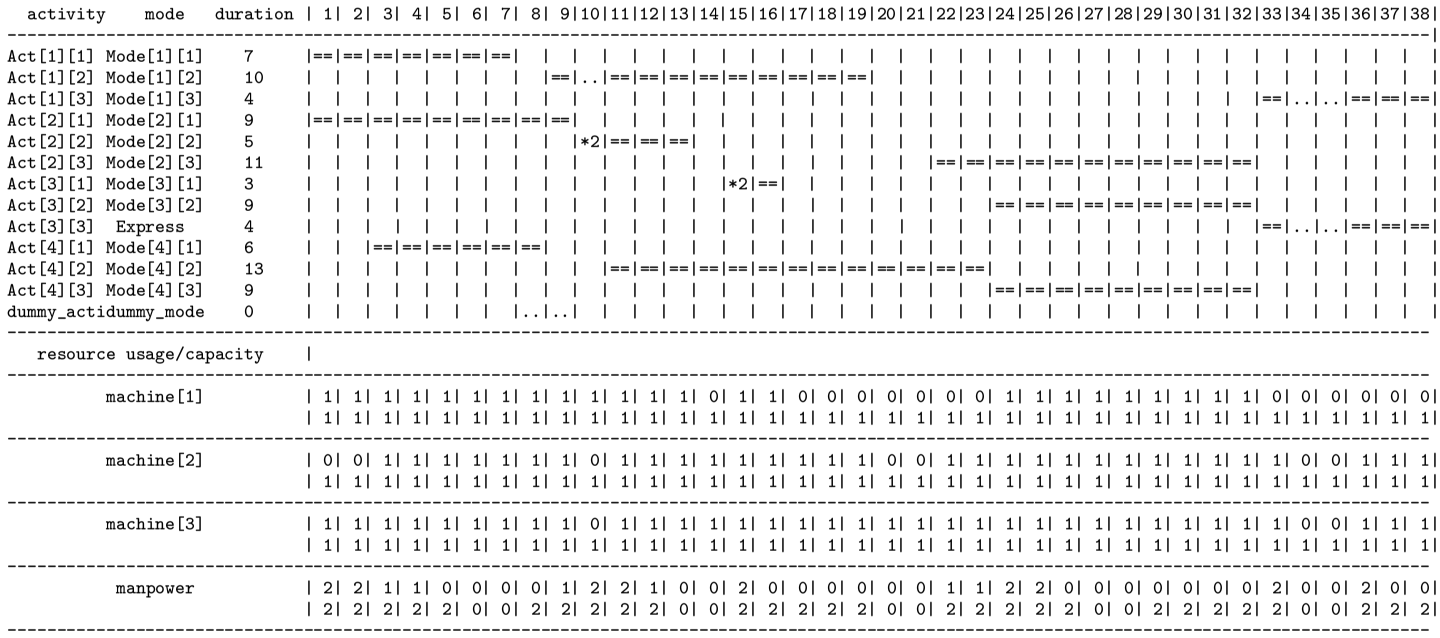

図 27: 例題のガントチャート表示．==は活動を処理中，. . は中断中，*2 は並列で処理中．

索 引