スケジューリング最適化ソルバー OptSeq II

optseq2.py モジュール使用法ガイド
(Python 言語からの呼び出し方法)

LOG OPT Co., Ltd.

ご注意

OptSeq II は，スケジューリング問題に特化した最適化ソルバーであり，実務における複雑な条件が付加されたスケジューリング問題に対して短時間で良好な解を探索することができる．OptSeq II の詳細については，`http://www.logopt.com/OptSeq/OptSeq.htm` を参照されたい．

ここでは，スケジューリング最適化ソルバー OptSeq II を，Python 言語から直接呼び出して求解するためのモジュール (`optseq2`) の使用法について解説する．

以下の構成は次の通り．

1 節では，OptSeq II で対象とする資源制約付きスケジューリング問題について述べる．

2 節では，OptSeq II に内在するオブジェクトについて解説する．

3 節では，最適化の動作をコントロールするためのパラメータについて述べる．

4 節では，オブジェクトに付随する属性について述べる．

5 節では，簡単な例を用いて OptSeq II の使用法を解説する．

1　資源制約付きスケジューリング問題

ここでは，OptSeq II で対象とする資源制約付きスケジューリング問題について簡単に解説する．詳しくは OptSeq II 導入ガイドを参照されたい．

行うべき仕事（ジョブ，作業，タスク）を**活動** (activity) とよぶ．スケジューリング問題の目的は活動をどのようにして時間軸上に並べて遂行するかを決めることであるが，ここで対象とする問題では活動を処理するための方法が何通りかあって，そのうち 1 つを選択することによって処理するものとする．このような活動の処理方法を**モード** (mode) とよぶ．納期や納期遅れのペナルティ（重み）は活動ごとに定めるが，作業時間や資源の使用量はモードごとに決めることができる．

活動を遂行するためには**資源** (resource) を必要とする．資源の使用可能量は時刻ごとに変化しても良いものとする．また，モードごとに定める資源の使用量も作業開始からの経過時間によって変化しても良いものとする．通常，資源は作業完了後には再び使用可能になるものと仮定するが，お金や原材料のように一度使用するとなくなってしまうものも考えられる．そのような資源を**再生不能資源** (nonrenewable resource) とよぶ．

活動間に定義される**時間制約** (time constraint) は，ある活動（先行活動）の処理が終了するまで，別の活動（後続活動）の処理が開始できないことを表す先行制約を一般化したものであり，先行活動の開始（完了）時刻と後続活動の開始（完了）時刻の間に以下の制約があることを規定する．

$$\text{先行活動の開始（完了）時刻} + \text{時間ずれ} \leq \text{後続活動の開始（完了）時刻}$$

ここで，納期ずれは任意の整数値であり負の値も許すものとする．この制約によって，活動の同時開始，最早開始時刻，時間枠などの様々な条件を記述することができる．

OptSeq II では，モードを作業時間分の小作業の列と考え，処理の途中中断や並列実行も可能であるとする．その際，中断中の資源使用量や並列作業中の資源使用量も別途定義できるものとする．また，時刻によって変化させることができる**状態** (state) が準備され，モード開始の状態の制限やモードによる状態の推移を定義できる．

2 オブジェクト

OptSeq II の Python モジュール (optseq2) におけるオブジェクト（クラス，インスタンス）には，以下のものがある．

- モデル Model
- 活動 Attribute
- モード Mode
- 資源 Resource
- 時間制約 Temporal
- 状態 State

本節では上の諸オブジェクトについて解説を行う．

注意：
OptSeq II では活動，モード，資源名を文字列で区別するため重複した名前を付けることはできない．なお，使用できる文字列は, 英文字 (a–z, A–Z), 数字 (0–9), 角括弧 ([ ]), アンダーバー (_), および @ に限定される．また，作業名は source,sink 以外，モードは dummy 以外の文字に限定される．

2.1 モデル

Python から OptSeq II を呼び出して使うときに，最初にすべきことはモデルクラスのインスタンスを生成することである．OptSeq II では引数なしで，以下のように記述する．

```
モデルインスタンス=Model()
```

モデルオブジェクトは，以下のメソッドをもつ．

活動追加メソッド (addActivity): モデルに 1 つの活動を追加する．返値は活動オブジェクトである．引数の名前と意味は以下の通り．

name: 活動の名前を文字列で与える．ただし活動の名前に"source","sink"を用いることはできない．

duedate: 活動の納期を 0 以上の整数もしくは，無限大"inf"で与える．省略可で，既定値は無限大"inf".

weight: 活動の完了時刻が納期を遅れたときの単位時間あたりのペナルティ．省略可で，規定値は 1.

資源追加メソッド (addResource): モデルに資源を 1 つ追加する．返値は資源オブジェクトである．引数の名前と意味は以下の通り．

name: 資源の名前を文字列で与える．

capacity: 資源の容量（使用可能量の上限）を辞書もしくは正数値で与える．正数値で与えた場合には，開始時刻は 0，終了時刻は無限大と設定される．辞書のキーはタプル (開始時刻 , 終了時刻) であり，値は容量を表す正数値である．開始時刻と終了時刻の組を**区間** (interval) とよぶ．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 と定義する．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．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」となる（図 1）．

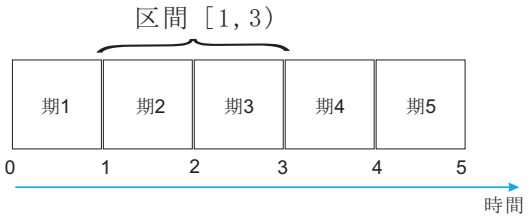

図 1: 区間の例.

rhs: 再生不能資源制約の右辺定数を与える．省略可で，規定値は 0.

direction: 再生不能資源制約の方向を示す文字列を与える．文字列は`"<="`, `">="` のいずれかとする．省略可であり，規定値は `"<="`.

時間制約追加メソッド (addTemporal): モデルに時間制約を 1 つ追加する．返値は時間制約オブジェクトである．時間制約は，先行活動と後続活動の開始（もしくは完了）時刻間の関係を表し，以下のように記述される．

$$\text{先行活動の開始（完了）時刻} + \text{時間ずれ} \leq \text{後続活動の開始（完了）時刻}$$

ここで**時間ずれ** (delay) は時間の差を表す整数値である．先行（後続）活動の開始時刻か完了時刻のいずれを対象とするかは，時間制約のタイプで指定する．タイプは，**開始時刻** (start time) のとき文字列`"S"`，**完了時刻** (completion time) のとき文字列`"C"`で表し，先行活動と後続活動のタイプを 2 つつなげて`"SS"`, `"SC"`, `"CS"`, `"CC"`のいずれかから選択する．引数の名前と意味は以下の通り．

pred: **先行活動** (predecessor) のオブジェクトもしくは文字列`"source"`を与える．文字列`"source"`は，すべての活動に先行する開始時刻 0 のダミー活動を定義するときに用いる．

succ: **後続活動** (successor) のオブジェクトもしくは文字列`"source"`を与える．文字列`"source"`は，すべての活動に先行する開始時刻 0 のダミー活動を定義するときに用いる．

tempType: 時間制約のタイプを与える．`"SS"`,`"SC"`,`"CS"`,`"CC"`のいずれかから選択し，省略した場合の規定値は`"CS"` （先行活動の完了時刻と後続活動の開始時刻）である．

delay: 先行活動と後続活動の間の時間ずれを整数値（負の値も許すことに注意）で与える．規定値は 0 である．

状態追加メソッド (addState): モデルに状態を追加する．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ある．

最適化メソッド (optimize): モデルの最適化を行う．返値はなし．最適化を行った結果は，活動，モード，資源，時間制約オブジェクトの属性に保管される．

ガントチャート出力メソッド (write): 最適化されたスケジュールを簡易**ガントチャート** (Gantt chart) [1] としてテキストファイルに出力する．引数はファイル名 (`filename`) であり，その規定値は `optseq.txt` である．

モデルオブジェクトは，モデルの情報を文字列として返すことができる．たとえば，`m1` と名付けたモデルオブジェクトの情報は，

```
print m1
```

で得ることができる．

2.2 活動

成すべき仕事（作業，タスク）を総称して**活動** (activity) とよぶ．活動オブジェクトは，モデルに含まれる形で生成される．活動インスタンスは，上述したモデルの活動追加メソッド (addActivity) の返値として生成される．

```
活動インスタンス=Model.addActivity(引数)
```

上のメソッドの引数については，2.1 節を参照．

活動には任意の数のモード（活動の実行方法）を追加することができる．モードの追加は，以下のメソッドで行う．

モード追加メソッド (addModes): 活動にモードを追加するメソッド．引数は（任意の数の）モードオブジェクト．

2.3 モード

活動の処理方法を**モード** (mode) とよぶ． 活動は少なくとも 1 つのモードをもち，そのうちのいずれかを選択して処理される．

モードのインスタンスは，モードクラス `Mode` から生成される．

```
モードインスタンス=Mode(引数)
```

引数の名前と意味は以下の通り．

name: モードの名前を文字列で与える．ただしモードの名前に`"dummy"`を用いることはできない．

duration: モードの作業時間を非負の整数で与える．規定値は 0.

autoselect: モードを自動選択するか否かを表すフラグ．モードを自動選択するとき `True`，それ以外のとき `False` を設定する．規定値は `False`.

[1] Henry Gantt によって 100 年くらい前に提案されたスケジューリングの表記図式.

モードオブジェクトは，以下のメソッドをもつ．

資源追加メソッド (addResource): モードを実行するときに必要な資源とその量を指定する．引数と意味は以下の通り．

resource: 追加する資源のオブジェクトを与える．

requirement: 資源の必要量を辞書もしくは正数値で与える．辞書のキーはタプル (開始時刻 , 終了時刻) であり，値は資源の使用量を表す正数値である．正数値で与えた場合には，開始時刻は 0，終了時刻は無限大と設定される．

rtype: 資源のタイプを表す文字列．`"break"`,`"max"` のいずれかから選択する．`"break"`を与えた場合には，中断中に使用する資源量を指定する．`"max"`を与えた場合には，並列処理中に使用する資源の最大量を指定する．省略可で，その場合には，並列処理中に資源使用量の総和を（並列で行った分だけ）使用する．

中断追加メソッド (addBreak): モードは単位時間ごとに分解された作業時間分の小作業の列と考えられる．小作業を途中で中断してしばらく時間をおいてから次の小作業を開始することを**中断** (break) とよぶ．中断追加メソッド (addBreak) は，モード実行時における中断の情報を指定する． 引数と意味は以下の通り．

start: 中断可能な最早時刻を与える．省略可で，規定値は 0.

finish: 中断可能時刻の最遅時刻を与える．省略可で，規定値は 0.

maxtime: 最大中断可能時間を与える．省略可で，規定値は無限大 (`"inf"`).

並列追加メソッド (addParallel): モードは単位時間ごとに分解された作業時間分の小作業の列と考えられる．資源量に余裕があるなら，同じ時刻に複数の小作業を実行することを**並列実行** (parallel execution) とよぶ． 並列追加メソッド (addParallel) は，モード実行時における並列実行に関する情報を指定する．

引数と意味は以下の通り．

start: 並列実行可能な最小の小作業番号を与える．省略可で，規定値は 1.

finish: 並列実行可能な最大の小作業番号を与える．省略可で，規定値は 1

maxparallel: 同時に並列実行可能な最大数を与える．省略可で，規定値は無限大 (`"inf"`).

状態追加メソッド (addState): 状態追加メソッド (addState) は，モード実行時における状態の値と実行直後（実行開始が時刻 t のときには，時刻 $t+1$）の状態の値を定義する．

引数と意味は以下の通り

state: モードに付随する状態オブジェクト．省略不可．

fromValue: モード実行時における状態の値．省略可で，規定値は 0.

toValue: モード実行直後における状態の値．省略可で，規定値は 0.

2.4 資源

資源オブジェクトは，モデルに含まれる形で生成される．資源インスタンスは，モデルの資源追加メソッド (addResource) の返値として生成される．

```
資源インスタンス=Model.addResource(引数)
```

上のメソッドの引数については，2.1 節を参照.

資源オブジェクトは，以下のメソッドをもつ..

容量追加メソッド (addCapacity): 資源の容量を区間と量の辞書に追加する．引数と意味は以下の通り.

start: 資源容量追加の開始時刻（区間の始まり）.

finish: 資源容量追加の終了時刻（区間の終わり）.

amount: 追加する資源量.

setRhs(右辺定数): 線形制約の右辺定数を設定する．引数は整数値（負の値も許す）とする.

setDirection(制約の向き): 制約の向きを設定する．引数は `"<="`, `">="` のいずれかとする.

addTerms(引数): 再生不能資源制約（の左辺）に 1 つもしくは複数の項を追加するメソッドである．活動がモードで実行されるときに 1，それ以外のとき 0 となる変数（値変数）を x[活動, モード] とすると，追加される項は,

$$\text{係数} \times x[\text{活動}, \text{モード}]$$

と記述される．`addTerms` メソッドの引数は以下の通り.

coeffs: 追加する項の係数もしくは係数リスト．係数もしくは係数リストの要素は整数（負の値も許す）.

vars: 追加する項の活動オブジェクトもしくは活動オブジェクトのリスト．リストの場合には，リスト `coeff` と同じ長さをもつ必要がある.

values: 追加する項のモードもしくはモードのリスト．リストの場合には，リスト `coeff` と同じ長さをもつ必要がある.

2.5 時間制約

時間制約オブジェクトは，モデルに含まれる形で生成される．時間制約インスタンスは，上述したモデルの時間制約追加メソッド (addTemporal) の返値として生成される.

```
時間制約インスタンス=Model.addTemporal(引数)
```

上のメソッドの引数については，2.1 節を参照.

2.6 状態

状態オブジェクトは，モデルに含まれる形で生成される．状態インスタンスは，上述したモデルの状態追加メソッド (addState) の返値として生成される．

```
状態インスタンス=Model.addState(引数)
```

状態オブジェクトは，指定時に状態の値を変化させるためのメソッド `addValue` をもつ．

addValue(引数): 引数は状態を変化させる時刻（`time`; 非負整数値）と変化後の値（`value`; 非負整数値）である．

3 パラメータ

OptSeq II に内在されている最適化ソルバーの動作は，パラメータ (parameter) を変更することによってコントロールできる．モデルオブジェクト `model` のパラメータを変更するときは，以下の書式で行う．

```
model.Params.パラメータ名 =値
```

たとえば，計算時間の上限 (`TimeLimit`) を 1 秒に変更したい場合には，

```
model.Params.TimeLimit=1
```

とする．

以下に代表的なパラメータとその意味を記す．

RandomSeed: 乱数系列の種を設定する．規定値は 1.

Makespan: 最大完了時刻（一番遅く終わる活動の完了時刻）を最小にするとき `True`，それ以外のとき（各活動に定義された納期遅れの重み付き和を最小にするとき）`False` を設定する．規定値は `False`.

TimeLimit: 最大計算時間 (秒) を設定する．規定値は 600.

OutputFlag: 計算の途中結果を出力させるためのフラグ．`True` のとき出力 On，`False` のとき出力 Off．規定値は `False`.

4 属性

モデル，活動，モード，資源，時間制約の情報は，オブジェクトの属性に保管されている．オブジェクトの属性は「オブジェクト.属性名」でアクセスできる．

たとえば，活動オブジェクト `act` の納期を `10`，重みを `2` に変更したい場合には，納期を表す属性が `duedate`，重みを表す属性が `weight` であるので，以下のようにすれば良い．

```
act.duedate=10
act.weight=2
```

モデルオブジェクトの代表的な属性は以下の通り.

activities: モデルに含まれる活動名をキー，オブジェクトを値とした辞書.

modes: モデルに含まれるモード名をキー，オブジェクトを値とした辞書.

resources: モデルに含まれる資源名をキー，オブジェクトを値とした辞書.

temporals: モデルに含まれる時間制約の先行作業名と後続作業名のタプルをキー，オブジェクトを値とした辞書.

Params: モデルのパラメータオブジェクト.

活動の代表的な属性は以下の通り.

name: 活動名.

duedate: 活動の納期．0 以上の整数もしく無限大`"inf"`.

weight: 活動の完了時刻が納期を遅れたときの単位時間あたりのペナルティ.

modes: 活動に付随するモードオブジェクトのリスト.

モードの代表的な属性は以下の通り.

name: モード名.

duration: モードの作業時間.

requirement: 通常の作業中の資源の必要量を表す辞書．辞書のキーはタプル（開始時刻 , 終了時刻）であり，値は資源の使用量を表す正数値.

breakable: 中断中の資源の必要量を表す辞書．辞書のキーはタプル（開始時刻 , 終了時刻）であり，値は資源の使用量を表す正数値.

parallel: 並列作業中の資源の最大量を表す辞書．辞書のキーはタプル（開始時刻 , 終了時刻）であり，値は資源の使用量を表す正数値.

autoselect: モードを自動選択するとき `True`，それ以外のとき `False` を設定する．規定値は `False`.

資源の代表的な属性は以下の通り.

name: 資源名.

capacity: 資源の容量（使用可能量の上限）を表す辞書．辞書のキーはタプル（開始時刻 , 終了時刻）であり，値は容量を表す正数値である.

rhs: 再生不能資源制約の右辺定数.

表 1: 4 活動 3 機械スケジューリング問題のデータ.

	作業 1	作業 2	作業 3
活動 1	機械 1 / 7 日	機械 2 / 10 日	機械 3 / 4 日
活動 2	機械 3 / 9 日	機械 1 / 5 日	機械 2 / 11 日
活動 3	機械 1 / 3 日	機械 3 / 9 日	機械 2 / 12 日
活動 4	機械 2 / 6 日	機械 3 / 13 日	機械 1 / 9 日

direction: 再生不能資源制約の方向.

terms: 再生不能資源制約の左辺を表す項のリスト．各項は (係数 , 活動オブジェクト , モードオブジェクト) のタプルである.

時間制約の代表的な属性は以下の通り.

pred: 先行活動のオブジェクト.

succ: 後続活動のオブジェクト

type: 時間制約のタイプを表す文字列．`"SS"`（開始，開始）,`"SC"`（開始，完了）,`"CS"`（完了，開始）,`"CC"`（完了，完了）のいずれかである.

delay: 時間制約の時間ずれを表す整数値.

5 例

例として，4 活動 3 機械のスケジューリング問題を考える．各活動はそれぞれ 3 つの子活動（これを以下では作業とよぶ）1, 2, 3 から成り, この順序で処理しなくてはならない．各作業を処理する機械, および処理日数は，表 1 の通り.

このように，活動によって作業を行う機械の順番が異なる問題は，**ジョブショップ** (job shop) とよばれ，スケジューリングモデルの中でも難しい問題と考えられている.

目的は最大完了時刻最小化とする．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仮定する.

1. 各作業の初めの 2 日間は作業員資源を必要とする操作がある．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.
2. 各作業は，1 日経過した後だけ，中断が可能.
3. 機械 1 での作業は，最初の 1 日は 2 個まで並列処理が可能.

4. 機械 2 に限り, 特急処理が可能．特急処理を行うと処理日数は 4 日で済むが, 全体で 1 度しか行うことはできない.

5. 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.

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 活動のスケジューリングモデルとして OptSeq II で記述できる.

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する．このとき，機械資源の容量（使用可能量の上限）を 1 と設定しておく.

```
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 名と設定する（ただし最初の日は月曜日と仮定する）.

```
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 に設定しておく.

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

次に，活動とモードに関する記述を行う.

まず，表 1 の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く.

```
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 を準備しておく（これは機械 2 に限定した処理で作業時間は 4 である）.

```
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 に保管する（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 に対して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する（11 から 13 行目）.

```
act={}
mode={}
for (i,j) in JobInfo:
    act[i,j]=m1.addActivity("Act[%s][%s]"%(i,j))
    mode[i,j]=Mode("Mode[%s][%s]"%(i,j),duration=JobInfo[i,j][1])
    mode[i,j].addResource(machine[JobInfo[i,j][0]],{(0,"inf"):1},"max")
    mode[i,j].addResource(manpower,{(0,2):1})
    mode[i,j].addBreak(1,1)
    if JobInfo[i,j][0]==1:
        mode[i,j].addParallel(1,1,2)
    if JobInfo[i,j][0]==2:
        act[i,j].addModes(mode[i,j],express)
        budget.addTerms(1,act[i,j],express)
    else:
        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く．

```
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となる．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る．

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 で中断可能と設定する．そして, 中断中, 資源「機械 1」を消費し続けるものと定義する．
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 つの制約を追加する．

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る．

```
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 秒で求解した後でガントチャートをファイル `chart1.txt` に出力する．

```
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）が表示される．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 まで行われる．

```
--- 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 2 のようになる．

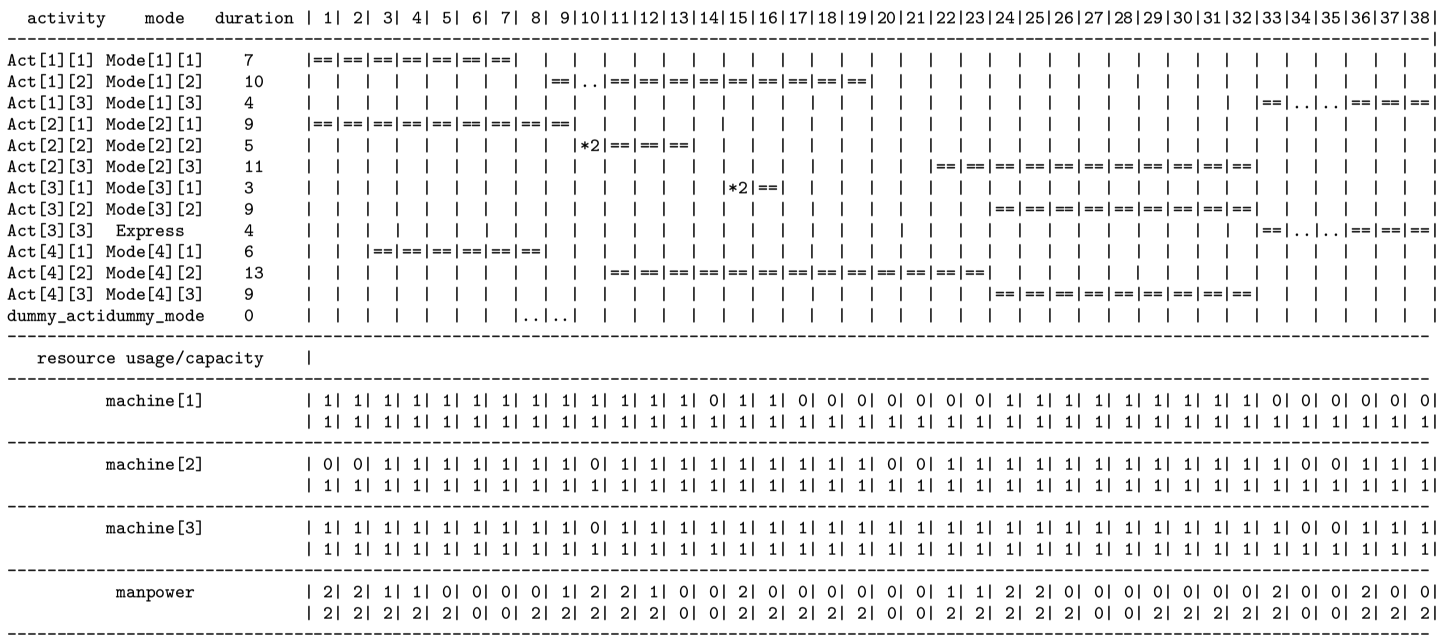

図 2: 例題のガントチャート表示．==は活動を処理中，. . は中断中，*2 は並列で処理中を表す．

索 引

activities, 9
addActivity, 3
addBreak, 6
addCapacity, 7
addParallel, 6
addResource, 3, 6
addState, 5
addTemporal, 4
addTerms, 7
addValue, 8

breakable, 9

capacity, 9

delay, 4, 10
direction, 4, 9
duedate, 9
duration, 9

Makespan, 8
Model, 3
modes, 9

optimize, 5
OutputFlag, 8

parallel, 9
Params, 9
pred, 4, 10

RandomSeed, 8
requirement, 9
resources, 9
rhs, 4, 9

setDirection, 7
setRhs, 7
succ, 4, 10

temporals, 9
tempType, 4
terms, 9
TimeLimit, 8
type, 10

variables, 9

weight, 9
write, 5

開始時刻 start time, 4
活動 activity, 2, 5
ガントチャート Gantt chart, 5
完了時刻 completion time, 4

期 period, 4

区間 interval, 4

後続活動 successor, 4

時間ずれ delay, 4
時間制約 time constraint, 2
資源 resource, 2
資源制約付きスケジューリング問題 resource constrained scheduling problem, 2
状態 state, 2
ジョブショップ job shop, 10

スケジューリング問題 scheduling problem
　資源制約付き— resource constrained—, 2

先行活動 predecessor, 4

属性 attribute, 8

中断 break, 6

パラメータ parameter, 8

並列実行 parallel execution, 6

モード mode, 2, 5
モード（コンストラクタ） Mode, 5
モデル（コンストラクタ） Model, 3