資源制約スケジューリング問題 II

1 問題定義

前節の「資源制約スケジューリング問題 I」と同様, 資源集合 $\mathcal{R} = \mathcal{R}^{\mathrm{re}} \cup \mathcal{R}^{\mathrm{non}}$, 作業集合 $\mathcal{J} = \{1, 2, \cdots, J\}$, および処理モード集合 $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_J$ が入力として与えられる. 再生可能資源制約, および再生不可能資源制約については前節と同様であるが, 以下の点で問題の定式化が異なる.

- 作業の中断・並列処理

  詳細は後述する. 作業の中断や並列処理を許可することで, より複雑なスケジューリング問題を記述することが可能となる.

- 時間制約

  先行制約の一般化として, 以下の不等式制約を扱うことができる:

  $$s_{j_1} + \delta_{ij}^{\mathrm{ss}} \leq s_{j_2}.$$

  ここで, s_{j_1}, s_{j_2} はそれぞれ作業 j_1, j_2 の開始時刻を表し, $\delta_{ij}^{\mathrm{ss}}$ の値は負であってもよい. また, s_{j_1}, s_{j_2} の代わりに, j_1, j_2 の完了時刻 c_{j_1}, c_{j_2} を用いた不等式制約

  $$s_{j_1} + \delta_{ij}^{\mathrm{sc}} \leq c_{j_2},$$

  $$c_{j_1} + \delta_{ij}^{\mathrm{cs}} \leq s_{j_2},$$

  $$c_{j_1} + \delta_{ij}^{\mathrm{cc}} \leq c_{j_2}$$

  を課すこともできる.

- 総納期遅れ最小化

  本定式化では, 各作業 $j \in \mathcal{J}$ には納期 $d_j \geq 0$ が与えられ, 総納期遅れ $\sum_j \max(0, c_j - d_j)$ を最小化することが目的である.

  前節とは異なり, 資源制約や時間制約はすべて絶対制約として扱われる.

なお, 本定式化に直前先行制約は陽には含まれないが, 以下に述べる「作業の中断」を用いることで同様の制約を扱うことが可能である.

作業の中断・並列処理

いま, 作業 $j \in \mathcal{J}$ が処理モード $m \in \mathcal{M}_j$ (処理時間 d_m) で処理されるとする. 便宜上, 作業 j は, それぞれ処理時間 1 の小作業 $j^1, j^2, \ldots, j^{d_m}$ からなるものと考える. さらに, 作業 j の開始, 完了を表す仮想的な (処理時間 0 の) 小作業 j^0, j^{d_m+1} を導入する (図 1 参照).

処理モード m は, 最大中断可能時間 $b_m(\tau)(\geq 0), \tau = 0, 1, \ldots, d_m$ を持ち, $b_m(\tau)$ の値が正であれば (0 であれば), 作業 j は, τ 番目と $\tau + 1$ 番目の小作業間で中断可能 (中断不可能) である.

また, 処理モード m は, 最大並列処理可能数 $p_m(\tau)(\geq 1), \tau = 1, 2, \ldots, d_m$ を持ち, $\tau' - \tau + 1 \leq p_m(\tau)$ であれば, τ 番目から τ' 番目までの (連続した $\tau' - \tau + 1$ 個) の小作業を並列して (すなわち 1 単位時間で) 処理することができる. すべての $\tau \in [1, d_m]$ に対して $p_m(\tau) = 1$ が成り立つとき, 作業 j は並列処理不可であることを意味する.

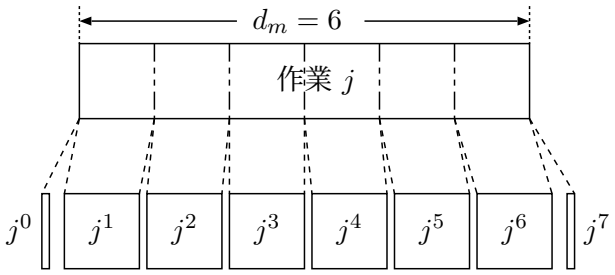


図 1: 小作業への分割

本定式化では, 各小作業 j^τ の再生可能資源消費量 $k_{m,r}(\tau)$ $(r \in \mathcal{R}_m)$ に加え, 作業中断中の資源消費量 $k^{\mathrm{b}}_{m,r}(\tau)$ を指定することもできる. すなわち $k^{\mathrm{b}}_{m,r}(\tau) > 0$ のとき, 作業 j は, τ 番目の小作業完了後, $\tau + 1$ 番目の小作業が開始されるまでの間, 資源 r を消費し続けることになる. ($b_m(\tau) = 0$ であれば, 中断不可であるため, $k^{\mathrm{b}}_{m,r}(\tau)$ の値は意味を持たない.)

並列処理が可能であれば ($\exists \tau, p_m(\tau) \geq 2$ であれば), 資源集合 $\mathcal{R}_m$ は $\mathcal{R}^{\mathrm{sum}}_m$ と $\mathcal{R}^{\mathrm{max}}_m$ に分割される. 複数の小作業 $j^\tau, \tau = \tau_1, \ldots, \tau_2$ が並列処理されるとき, $r \in \mathcal{R}^{\mathrm{sum}}_m$ に対しては総消費量

$$\sum_{\tau=\tau_1}^{\tau_2} k^{\mathrm{b}}_{m,r}(\tau)$$

が必要であるのに対し, $r \in \mathcal{R}^{\mathrm{max}}_m$ に対しては,

$$\max_{\tau \in [\tau_1, \tau_2]} k^{\mathrm{b}}_{m,r}(\tau)$$

のみが必要となる.

2 アルゴリズム

アルゴリズムの基本的な考え方は前節と同様であり, 全体の枠組みとしてはタブー探索を用いている. 各作業の処理モードを表すベクトル $\boldsymbol{m}$ と作業リスト $\boldsymbol{l}$ の組を解として, 各解から, リストスケジューリングによりスケジュールが生成される. ただし本定式化では, 作業の中断および並列処理を許しているため, リストスケジューリングの計算がやや煩雑になっている. さらに, 制約を満たすため, 作業リストにしたがって各作業の開始時刻を決定していく際, 一旦決定した作業の開始時刻を変更する (遅らせる) ということがしばしば必要になる (バックトラック). 一般に, リストスケジューリングの計算時間は入力サイズの多項式時間では抑えられないため, 本アルゴリズムでは, リストスケジューリング 1 回あたりのバックトラック回数に上限を設けている.

本アルゴリズムでは, タブー探索の対象となる処理モードベクトル $\boldsymbol{m}$ を, すべての再生不可能資源制約を満たすという意味で実行可能なもののみに限定している. 再生不可能資源制約を持たす $\boldsymbol{m}$ を求める問題は制約充足問題 (CSP) として記述できるので, 近傍操作で処理モードベクトル $\boldsymbol{m}$ を変更した際, $\boldsymbol{m}$ が実行不可能になった場合には, 4.1 節の CSP ソルバをサブルーチンとして用いることで $\boldsymbol{m}$ の実行可能性を保っている.

3 例題

前節の例題に以下の修正を加えた問題を考える.

- 各作業の最初 2 日間 (人的資源を必要とする部分) は, 中断可能である.

- 機械 1 に限り, (人的資源に余裕があれば) 最初の 2 日間分の作業を並列処理することもできる.

最大完了時刻最小化は, すべての作業に後続する仮想作業 sink の納期を 0, それ以外の作業に対しては納期を ∞ とすることで実現できる.

また, 本定式化では直前先行制約を直接用いることはできないが,

- 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,

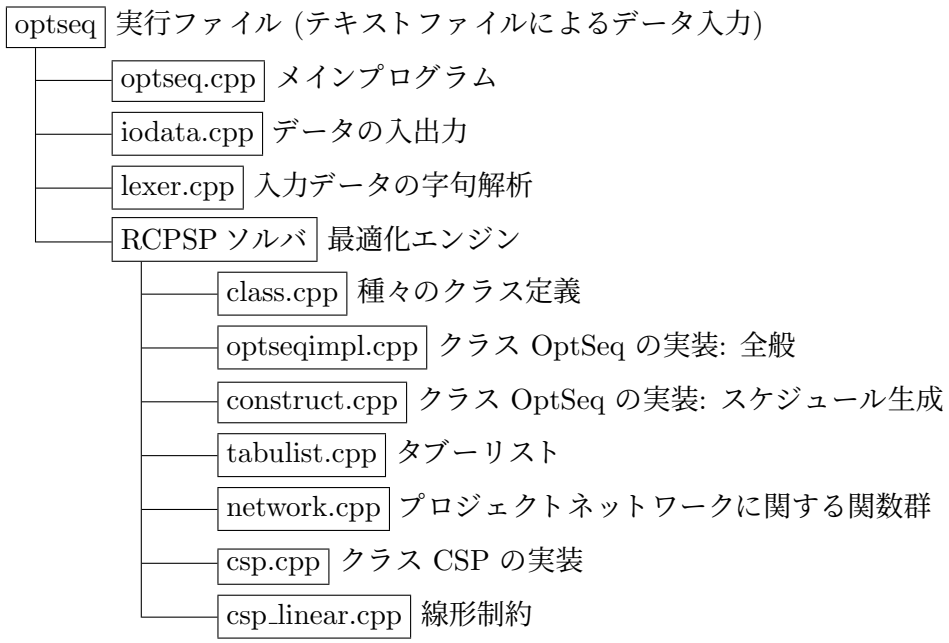
といった制約は以下のようにして扱うことができる.

1. 処理時間 0 の仮想作業 dummy を導入し, $\tau = 0$ で中断可能とする. 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, (仕事 1・作業 1 の完了時刻) = (dummy の開始時刻) および (dummy の完了時刻) = (仕事 2・作業 2 の開始時刻) の 2 つの等式制約を追加する.

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.

4 プログラムの構成

プログラムの構成を以下に示す.



本プログラムのエンジンである RCPSP ソルバは C++ クラスライブラリであり, プログラム optseq は, テキストファイルからデータ入力できるよう簡易インタフェース (iodata.cpp, lexer.cpp) を加えたものである.

5 プログラムの使用法

make コマンドでコンパイル後, コンソール上で

```
% optseq -help
```

とすれば

```
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
```

と出力される. まず, これらのオプションについて説明する.

`-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 (詳細は省略する).

`-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.

問題例のデータは標準入力から読み込まれる. データを記述した入力ファイルを *inputfile* として,

`optseq <` *inputfile*

で実行できる. 以下, 入力フォーマットについて述べる.

入力ファイルフォーマット

入力フォーマットは,

1. 資源
2. 処理モード・作業
3. 時間制約

の定義からなる. 未定義の資源名, 処理モード名, 作業名を用いない限り, 記述順序に制限はない. なお, # 以降その行末まではコメントとして無視される. また, 改行はコメントの終了を表す以外に意味を持たない.

1. 資源

```
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
       ...
```

[時刻 1, 時刻 2) で与えられる区間ごとに, 資源供給量を指定する. 記述のない区間は供給量 0 (資源使用不可) と見なされる.

2. 処理モード・作業

```
mode  処理モード名  duration  処理時間
```

で処理モード名と処理時間を指定した後, 必要に応じて以下の記述を追加する.

- 中断可能区間:

  ```
  break interval 開始  終了  max 最大中断時間
        interval 開始  終了  max 最大中断時間
        ...
  ```

  中断可能区間, および最大中断可能時間 $b_m(\tau)$ を指定する. 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` と見なされる. 例えば

  ```
  break interval 2 4
        interval 8 8
  ```

  は, 2, 3, 4 または 8 番目の小作業を終えた時点でのみ中断可能であることを意味する.

- 並列処理可能区間:

  ```
  parallel interval 開始  終了  max 最大並列数
           interval 開始  終了  max 最大並列数
           ...
  ```

  並列処理可能区間, および最大並列処理可能数 $p_m(\tau)$ を指定する. 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` と見なされる. 例えば

  ```
  break interval 2 4 max 4
  ```

  は, 2 番目, 3 番目, 4 番目の小作業から, それぞれ最大 4 個の小作業を並列処理可能であることを意味する.

- 資源使用量:

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

  指定区間の資源使用量を与える. 区間は,「開始　終了」もしくは「`break`　開始　終了」のいずれかである. 後者により, 処理中断中の資源使用量を指定することができる. 記述のない区間は, 使用量 0 と見なされる. 例えば,

```
res  interval 5 8  requirement 1
     break interval 3 3 requirement 1
```

は, 処理開始後 5 単位時間後から 8 単位時間後まで (すなわち, 6, 7, 8 番目の小作業処理中), および, 3 番目の小作業完了後, 4 番目の小作業開始までの間, 資源 res を 1 単位ずつ使用することを意味する.

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るが,

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

とすることで, 資源使用量を総和ではなく, 最大量とすることができる.

作業の定義は

```
activity  作業名  duedate 納期
   処理モード名
   処理モード名
   ...
```

で行われる. 納期は省略可能であり, その場合, duedate inf と見なされる. 処理モードは 1 つ以上であれば, 数に制限はない. また, 選択可能な処理モードが 1 つだけであれば,

```
activity  作業名  duedate 納期
   mode  duration  ...
```

として, activity の記述内で mode を記述することができる (この場合, 処理モード名は不要). なお, source および sink はデフォルトで定義されている. 最大完了時刻最小化を目的関数とする場合,

```
activity sink duedate 0
```

を追加すればよい.

3. 時間制約

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

(先行作業の開始もしくは完了時刻)+ (時間ずれ) $\leq$ (後続作業の開始もしくは完了時刻) を表す. ここで, type は SS, SC, CS, CC のいずれかであり, 例えば type SS は, (開始時刻) + (時間ずれ) $\leq$ (開始時刻) の形の制約を意味する. なお, type および delay の記述は省略可能であり, その場合, それぞれ type CS, delay 0 と見なされる.

4. 再生不可能資源制約

```
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 と見なされる. 消費量, 供給量は負の値であってもよい.

3 章の問題例を記述した入力ファイル (一部省略) を以下に示す.

```
# 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

# 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

# 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] interval 0 4  requirement 1
     manpower   interval 0 2  requirement 1

# 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

mode mode[1][2] duration 10
     break interval 1 1
     machine[2] interval 0 10 requirement 1
     manpower   interval 0 2  requirement 1
activity activity[1][2] mode[1][2]  express


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

（中略）

```
# 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]

# 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


# 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

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

これを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が表示される. 例えば, activity[1][1] の開始時刻は 2 であり, まず時刻 2〜3 の間に 2 つの小作業が並列処理された後 (2--3[2]), 残りの小作業が時刻 8 まで行われている. また, activity[1][2] の処理モードは mode[1][2] であり, 時刻 8〜9 に処理が行われた後, 中断を挟み, 時刻 10 に処理が再開される.

```
% optseq -time 1 < sample
# reading data ... done: 0.01(s)
# random seed: 1
```

```
# tabu tenure: 0
# cpu time limit: 1.00(s)
# iteration limit: 1073741823
# computing all-pairs longest paths and strongly connected components ... done
#scc 13
objective value = 60(cpu time = 0.00(s), iteration = 0)
0: 0.00(s): 60/60
objective value = 53(cpu time = 0.00(s), iteration = 1)
objective value = 46(cpu time = 0.00(s), iteration = 2)
objective value = 43(cpu time = 0.00(s), iteration = 3)
objective value = 41(cpu time = 0.02(s), iteration = 19)
objective value = 39(cpu time = 0.02(s), iteration = 20)
objective value = 38(cpu time = 0.03(s), iteration = 21)

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

--- best solution ---
source ---: 0 0
sink ---: 38 38
activity[1][1] ---: 2 2--3[2] 3--8 8
activity[1][2] mode[1][2]: 8 8--9 10--19 19
activity[1][3] ---: 32 32--33 35--38 38
activity[2][1] ---: 0 0--9 9
activity[2][2] ---: 9 9--10[2] 10--13 13
activity[2][3] mode[2][3]: 21 21--32 32
activity[3][1] ---: 14 14--15[2] 15--16 16
activity[3][2] ---: 23 23--32 32
activity[3][3] express: 32 32--33 35--38 38
activity[4][1] mode[4][1]: 0 0--6 6
activity[4][2] ---: 10 10--23 23
activity[4][3] ---: 23 23--32 32
act[1][1]_[2][2] ---: 8 9

objective value = 38
cpu time = 0.03/1.00(s)
iteration = 21/871
```

参考文献

[1] K. Nonobe and T. Ibaraki, A tabu search algorithm for a generalized resource constrained project scheduling problem, The Fifth Metaheuristics International Conference (MIC2003), pp. 55-1–55-6, 2003.