06

第6章

スケジューリング問題

1機械・フローショップ・資源制約付き問題の様々な定式化

この章の内容

1. 1機械リリース時刻付き重み付き完了時刻和問題
2. 1機械総納期遅れ最小化問題
3. 順列フローショップ問題
4. 資源制約付きスケジューリング問題

6.1

1機械リリース時刻付き重み付き完了時刻和問題

材料の到着日が違う6つの特注品を、どの順に作れば良いか？

例題と問題の定義

- 6つの得意先から特注品の製造を依頼された
  - 製造日数 1,4,2,3,1,4 日、材料の到着は 4,0,2,4,1,5 日後
  - 重要度は 3,1,2,3,1,2
- 単一の機械で n 個のジョブを処理する
  - 一度に1つのジョブしか処理できず、中断も不可
- 処理時間 p[j]・重み w[j]・リリース時刻 r[j] が与えられる
- 完了時刻 C[j] の重み付き和を最小にする順番を求める

! ポイント

得意先をジョブ、材料の到着日をリリース時刻、製造日数を処理時間、重要度を重みとみなす。

例題のデータ

1機械リリース時刻付き重み付き完了時刻和問題の例題

ジョブ j	1	2	3	4	5	6
処理時間 p[j]	1	4	2	3	1	4
リリース時刻 r[j]	4	0	2	4	1	5
重み w[j]	3	1	2	3	1	2

*WSPT*ルールでは目的関数値 *119*、最早リリース時刻ルールでは *110* となる

優先ルールと最適解

小規模な例でも、優先ルールが最適解を与えるとは限らない

1 WSPTルール：119

重みを処理時間で除した値 w[j]/p[j] の大きい順に前詰めで並べる

2 最早リリース時刻ルール：110

リリース時刻の早い順に前詰めで並べる

3 定式化による最適解：89

離接定式化ならびに完了時刻定式化により得られる解

離接定式化

s_j：ジョブ j の開始時刻、x_{jk}：ジョブ j が k に先行するとき 1

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} w_j s_j + \sum_{j=1}^{n} w_j p_j & \\
\text{subject to} \quad & s_j + p_j - M(1 - x_{jk}) \le s_k & \forall j \neq k \\
& x_{jk} + x_{kj} = 1 & \forall j < k \\
& s_j \ge r_j & \forall j = 1, \ldots, n \\
& x_{jk} \in \{0, 1\} & \forall j \neq k
\end{aligned}
$$

- 目的関数は Σ w[j](s[j]+p[j]) を展開したもの（第2項は定数）
- x[j,k]=1 なら j の完了後に k が開始
- x[j,k]+x[k,j]=1 が離接制約（名前の由来）
- M は max r[j] + Σ p[j] とすれば十分

離接定式化のAMPLモデル

disjunctive.mod

```
set J ordered;                  # ジョブの集合

param p {J} > 0;                # ジョブ j の処理時間
param r {J} >= 0;               # ジョブ j のリリース時刻
param w {J} >= 0;               # ジョブ j の重み
param M = (max {j in J} r[j]) + sum {j in J} p[j];

var s {j in J} >= r[j];         # ジョブ j の開始時刻
var x {j in J, k in J: j <> k} binary;

minimize Cost:
   sum {j in J} w[j] * s[j];

subject to Disjunctive {j in J, k in J: j <> k}:
   s[j] - s[k] + M*x[j,k] <= M - p[j];

subject to Pair {j in J, k in J: ord(j) < ord(k)}:
   x[j,k] + x[k,j] = 1;
```

- M は定義式のまま宣言（Python 側の計算不要）
- s の宣言時に s[j] ≥ r[j] を付加
- ord を使うには set J ordered が必要

big-Mを使わずに書く

disjunctive_mp.mod

```
set J ordered;

param p {J} > 0;
param r {J} >= 0;
param w {J} >= 0;
param M = (max {j in J} r[j]) + sum {j in J} p[j];

var s {j in J} >= r[j], <= M;

minimize Cost:
   sum {j in J} w[j] * s[j];

subject to NoOverlap
   {j in J, k in J: ord(j) < ord(k)}:
   s[k] >= s[j] + p[j] or s[j] >= s[k] + p[k];
```

- 「j が先か k が先か」を or でそのまま書く
- MP層が同等の線形の式へ自動で書き換える
- 論理式中の変数には上下限が必須（<= M）

欄外：離接制約と論理条件

2本の制約のいずれかが成立する条件は、0-1変数 y と大きな数 M で書ける

$$\text{いずれかが成立：} \quad 2x_1 + x_2 \leq 30 + My, \quad x_1 + 2x_2 \leq 20 + M(1-y)$$

$$\text{if A then B：} \quad (x_1 + x_2 > 10) \text{ or } (x_1 \leq 5)$$

$$x_1 + x_2 > 10 - My, \quad x_1 \leq 5 + M(1-y)$$

- y=0 なら1本目、y=1 なら2本目が有効になる
- if A then B は NOT A or B と同値（真偽表で確認できる）
- AMPLでは or を用いて簡潔に書ける

Smithの定理とWSPTルール

- リリース時刻のない1機械重み付き完了時刻和問題を考える
- w[j]/p[j] の非増加順に並べたスケジュールの中に最適解がある
- 証明：順序が逆の隣接ジョブ j,k を入れ替える
  - 目的関数の差は p[j]p[k](w[j]/p[j] − w[k]/p[k]) < 0
- この証明法は交換原理と呼ばれる
- w[j]/p[j] の非増加順に並べる解法が WSPTルール

! ポイント

交換原理は、スケジューリング問題が適当な指標の並べ替えで解けることを示す際の常套手段である。

1機械スケジューリング問題の妥当不等式

$j \in S$ では $w_j = p_j$、それ以外では $w_j = 0$ とした問題にSmithの定理を適用する

$$\sum_{j \in S} p_j C_j \geq \frac{1}{2} \sum_{j \in S} p_j^2 + \frac{1}{2} \left(\sum_{j \in S} p_j \right)^2 \quad \forall S \subseteq J$$

- S のジョブを先に処理するスケジュールが最適になることから導かれる
- この不等式から構成される多面体は1機械スケジューリング問題の多面体と一致する（Queyranne）
- 不等式の数は非常に多く、すべてを事前に列挙するのは実用的でない

分離問題を解くアルゴリズム

完了時刻 $(C_1, \ldots, C_n)$ を与え、破られた妥当不等式を1つ返すか、すべて満たすと判定する

1 1. 並べ替え

ジョブを C[j] の非減少順に 1,2,...,n と並べ替える

2 2. 部分集合

部分集合 $S_k = \{1, 2, \ldots, k\}$ を定義する

3 3. 最大化

f(S) = ½Σp[j]² + ½(Σp[j])² − Σp[j]C[j] を最大にする k* を求める

4 4. 判定

$f(S_{k^*}) \le 0$ ならすべて満たす。正なら S_{k^*} の不等式が破られている

完了時刻定式化

完了時刻 C_j を変数とし、妥当不等式を必要に応じて追加する

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} w_j C_j \\
\text{subject to} \quad & \sum_{j \in S} p_j C_j \geq \frac{1}{2} \sum_{j \in S} p_j^2 + \frac{1}{2} \left(\sum_{j \in S} p_j \right)^2 && \forall S \subseteq \{1, \ldots, n\}, S \neq \emptyset \\
& C_j \geq r_j + p_j && \forall j = 1, \ldots, n
\end{aligned}
$$

- 最初の制約は妥当不等式（分離アルゴリズムで必要なものだけ追加）
- 2番目の制約：完了時刻はリリース時刻＋処理時間以降

切除平面法のAMPLモデル

sched_cut.mod

```
set J ordered;

param p {J} > 0;
param r {J} >= 0;
param w {J} >= 0;

param nCut >= 0 integer, default 0;
set CUTS = 1..nCut;
set SS {CUTS} within J;

var C {j in J} >= r[j] + p[j];   # ジョブ j の完了時刻

minimize Cost:
   sum {j in J} w[j] * C[j];

subject to Cut {k in CUTS}:
   sum {j in SS[k]} p[j] * C[j]
      >= (sum {j in SS[k]} p[j]^2) / 2
         + (sum {j in SS[k]} p[j])^2 / 2;
```

- 切除平面の本数を nCut で保持
- 各切除平面のジョブ集合 S を添え字付き集合 SS で保持
- 制約 Cut は妥当不等式をそのまま書き写したもの

切除平面法による近似解

- 初期の切除平面として全ジョブの集合 J の不等式を与える
- 解く → 完了時刻順に並べる → $f(S_k)$ 最大の k^* を探す
- f の最大値が正なら S_{k^*} の不等式を1本追加して再度解く
- 得られるのは線形最適化問題（緩和問題）の解
- 緩和解の順にジョブを並べた解を近似解とする

例題の結果

例題では緩和解の順はジョブ $5,3,1,4,6,2$ となり、目的関数値 89 で最適解と一致する。ただし一般には最適とは限らない。

時刻添え字定式化

X_{jt}：ジョブ j が期 t に開始するとき 1、費用は $C_{jt} = w_j(t-1+p_j)$

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} C_{jt} X_{jt} \\
\text{subject to} \quad & \sum_{t=r_j+1}^{T-p_j+1} X_{jt} = 1 && \forall j = 1, \ldots, n \\
& \sum_{j} \sum_{t'=t-p_j+1}^{t} X_{jt'} \leq 1 && \forall t = 1, \ldots, T \\
& X_{jt} \in \{0, 1\} && \forall j, t
\end{aligned}
$$

- 時刻 t-1 から t までを期 t と呼ぶ
- 1番目：各ジョブを必ず1度開始する
- 2番目：同時に2つ以上のジョブを処理できない

時刻添え字定式化のAMPLモデル

time_index.mod

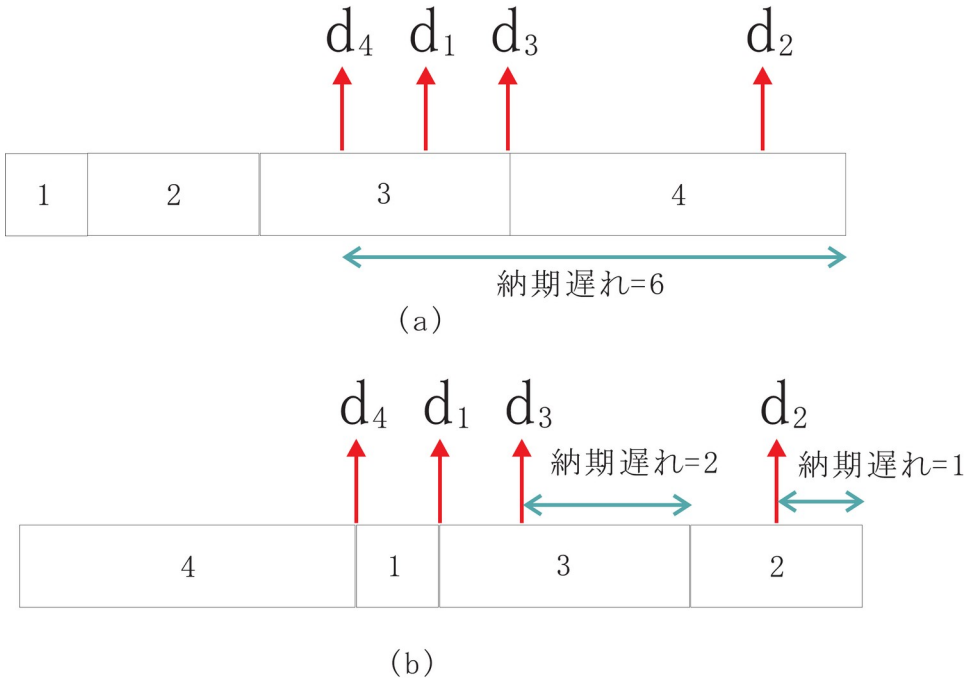
```
set J ordered;

param p {J} > 0;
param r {J} >= 0;
param w {J} >= 0;
param T = (max {j in J} r[j]) + sum {j in J} p[j];

set TX = {j in J, t in r[j]+1..T-p[j]+1};   # 開始しうる期の組

var X {TX} binary;    # ジョブ j を期 t に開始するとき 1

minimize Cost:
   sum {(j,t) in TX} w[j] * (t - 1 + p[j]) * X[j,t];

subject to Assign {j in J}:
   sum {t in r[j]+1..T-p[j]+1} X[j,t] = 1;

subject to NoOverlap {t in 1..T}:
   sum {j in J, t2 in max(t-p[j]+1, 1)..t: (j,t2) in TX}
      X[j,t2] <= 1;
```

- 集合 TX：意味のある (j,t) の組だけ
- 変数を TX 上で定義し必要な変数だけ作る
- NoOverlap：期 t に処理中のジョブは高々1つ

6.2 1機械総納期遅れ最小化問題

4社から依頼された原稿を、どの順に書けばペナルティが最小になるか？

売れっ子作家の原稿執筆



(a) 1,2,3,4 の順（総納期遅れ *6*）　*(b) 4,1,3,2* の順（総納期遅れ *3*、最適）

- A〜D の4社の原稿
  - 執筆日数 1,2,3,4 日
  - 納期 5,9,6,4 日後
- 1日遅れるごとに1万円のペナルティ
- 重み付き総納期遅れの最小化
  - NP困難である（Du–Leung）

時刻添え字定式化（総納期遅れ）

6.1節の時刻添え字定式化の費用 C_{jt} を定義し直すだけで良い

$$C_{jt} = w_j(t-1+p_j) \quad \longrightarrow \quad C_{jt} = w_j\,[t-1+p_j-d_j]^+$$

- $[\cdot]^+$ は $\max\{\cdot, 0\}$ を意味する
- 期 t に開始したときの完了時刻 t-1+p[j] と納期 d[j] の差が遅れ
- 制約は重み付き完了時刻和の場合と同じ

線形順序付け定式化

T_j：ジョブ j の納期遅れ、x_{jk}：j が k に先行するとき 1（big-M を用いない）

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j} w_j T_j \\
\text{subject to} \quad & x_{jk} + x_{kj} = 1 && \forall j < k \\
& x_{jk} + x_{k\ell} + x_{\ell j} \le 2 && \forall j \neq k \neq \ell \\
& \sum_{k \neq j} p_k x_{kj} + p_j \le d_j + T_j && \forall j = 1, \ldots, n \\
& x_{jk} \in \{0, 1\} && \forall j \neq k
\end{aligned}
$$

- 1番目は離接定式化と同じ
- 2番目：j→k→ℓ→j の巡回を禁止
- 1・2番目でジョブ上に線形順序が定まる
- 3番目：完了時刻が納期を超えた分が T[j]

線形順序付け定式化のAMPLモデル

linear_ordering.mod

```
set J ordered;

param p {J} > 0;               # 処理時間
param d {J} >= 0;              # 納期
param w {J} >= 0;              # 重み

var Tard {J} >= 0;             # ジョブ j の遅れ
var x {j in J, k in J: j <> k} binary;

minimize Cost:
   sum {j in J} w[j] * Tard[j];

subject to Tardiness {j in J}:
   sum {k in J: k <> j} p[k] * x[k,j] - Tard[j]
      <= d[j] - p[j];

subject to Pair {j in J, k in J: ord(j) < ord(k)}:
   x[j,k] + x[k,j] = 1;

subject to Triangle {j in J, k in J, l in J:
   ord(j) < ord(k) and ord(j) < ord(l) and k <> l}:
   x[j,k] + x[k,l] + x[l,j] <= 2;
```

6.3 順列フローショップ問題

3人の料理人の流れ作業で、特製おせちをなるべく早く仕上げるには？

特製おせちの例題

切り（機械1）→ 煮込み（機械2）→ 飾り付け（機械3）の流れ作業（単位は時間）

ジョブ j	1	2	3
機械1での処理時間 p[1,j]	2	4	1
機械2での処理時間 p[2,j]	3	2	4
機械3での処理時間 p[3,j]	1	3	1

1,2,3 の順では *13* 時間かかるが、*3,2,1* の順なら *11* 時間で終わる（最適）。各機械でジョブの処理順序は同一とする。

位置データ定式化

$x_{j\kappa}$：ジョブ j が κ 番目に投入されるとき 1、$s_{i\kappa}, f_{i\kappa}$：機械 i の κ 番目の開始・完了時刻

$$
\begin{aligned}
\text{minimize} \quad & f_{mn} \\
\text{subject to} \quad & \textstyle\sum_{\kappa} x_{j\kappa} = 1 && \forall j \\
& \textstyle\sum_{j} x_{j\kappa} = 1 && \forall \kappa \\
& f_{i\kappa} \le s_{i,\kappa+1} && \forall i;\ \kappa < n \\
& s_{i\kappa} + \textstyle\sum_{j} p_{ij} x_{j\kappa} \le f_{i\kappa} && \forall i, \kappa \\
& f_{i\kappa} \le s_{i+1,\kappa} && \forall i < m;\ \kappa \\
& x_{j\kappa} \in \{0,1\};\ s, f \ge 0
\end{aligned}
$$

- 各ジョブは1つの位置、各位置に1つのジョブ
- κ 番目の完了後に κ+1 番目が開始
- x と s, f を繋ぐ処理時間の制約
- 機械 i の完了後に機械 i+1 で開始

順列フローショップ問題のAMPLモデル

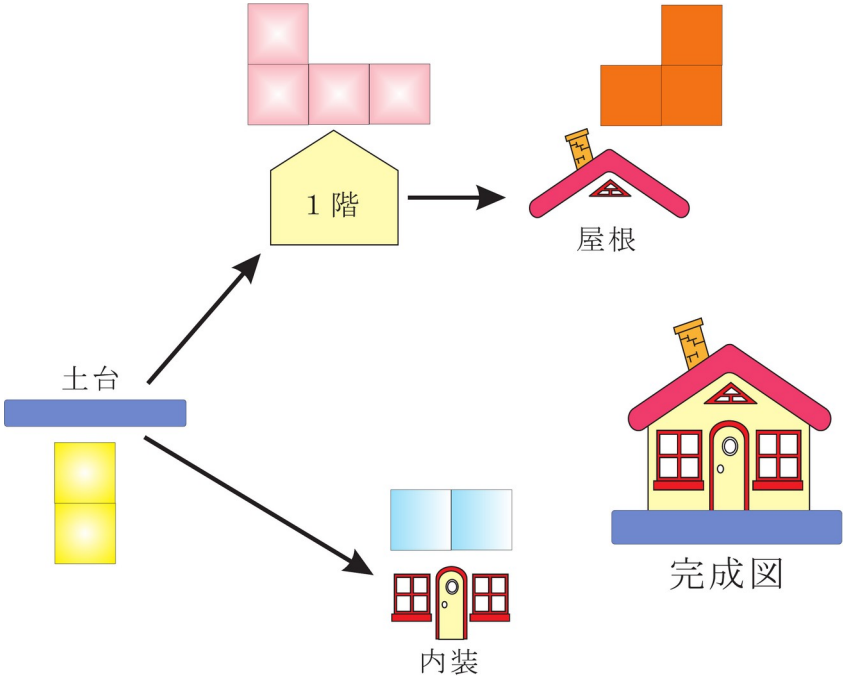
flow_shop.mod

```
param n > 0 integer;           # ジョブ数
param m > 0 integer;           # 機械数
set Jobs = 1..n;
set Machines = 1..m;
param p {Machines, Jobs} >= 0; # 機械 i でのジョブ j の処理時間

var x {Jobs, Jobs} binary;     # ジョブ j を k 番目に処理するとき 1
var s {Machines, Jobs} >= 0;   # 機械 i の k 番目の処理の開始時刻
var f {Machines, Jobs} >= 0;   # 機械 i の k 番目の処理の完了時刻

minimize Makespan: f[m,n];

subject to AssignJob {j in Jobs}: sum {k in Jobs} x[j,k] = 1;
subject to AssignPos {k in Jobs}: sum {j in Jobs} x[j,k] = 1;
subject to NextPos {i in Machines, k in Jobs: k <> n}:
   f[i,k] <= s[i,k+1];
subject to NextMachine {i in Machines, k in Jobs: i <> m}:
   f[i,k] <= s[i+1,k];
subject to Duration {i in Machines, k in Jobs}:
   s[i,k] + sum {j in Jobs} p[i,j] * x[j,k] <= f[i,k];
```

6.4 資源制約付きスケジューリング問題

作業員2人で、1階建てのお家をなるべく早く完成させるには？

お家を造る大工さんの例題



お家を造るための作業、先行制約、必要な人数

- 土台：2人で1日
- 1階の壁：1日目は2人、その後2日間は1人
- 内装：1人で2日
- 屋根：1日目は1人、次の1日は2人
- 作業員は2人、ただし3日目は1人

資源制約付きスケジューリング問題の定式化

x_{jt}：ジョブ j を期 t に開始するとき 1、a_{jrt}：開始 t 期後の資源 r の使用量

$$
\begin{aligned}
\text{minimize} \quad & \textstyle\sum_{j\in\mathcal{J}} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt} \\
\text{subject to} \quad & \textstyle\sum_{t=1}^{T-p_j+1} x_{jt} = 1 && \forall j \in \mathcal{J} \\
& \textstyle\sum_{j\in\mathcal{J}} \sum_{s=\max\{t-p_j+1,1\}}^{\min\{t,T-p_j+1\}} a_{jr,t-s} x_{js} \le RUB_{rt} && \forall r \in \mathcal{R}, t \in \mathcal{T} \\
& \textstyle\sum_{t=2}^{T-p_j+1} (t-1) x_{jt} + p_j \le \sum_{t=2}^{T-p_k+1} (t-1) x_{kt} && \forall (j,k) \in \mathcal{P} \\
& x_{jt} \in \{0,1\}
\end{aligned}
$$

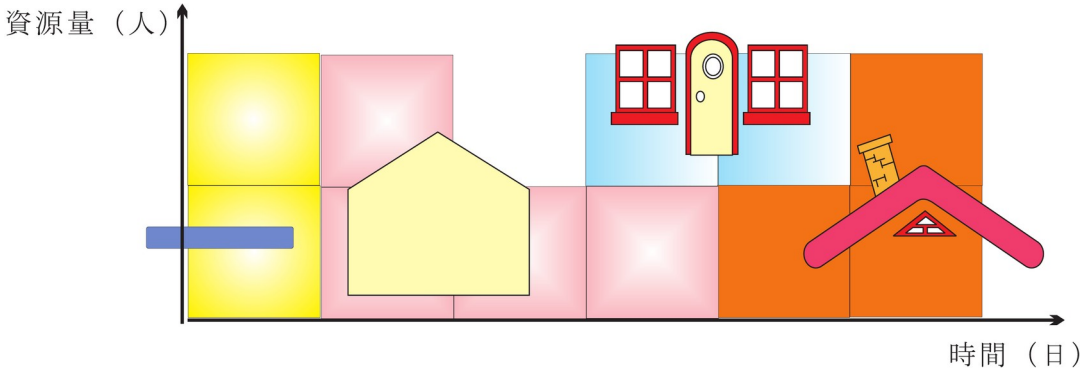
- ジョブ遂行制約：すべてのジョブを必ず1度処理
- 資源制約：期 t に作業中のジョブの使用量の合計が上限 RUB[r,t] 以下
- 時間制約（先行制約）：j の完了時刻 ≤ k の開始時刻

資源制約付きスケジューリングのAMPLモデル

rcs.mod

```
set J;                          # 作業の集合
set PR within {J,J};            # 先行関係 (j,k): j は k より前
set R;                          # 資源の集合
param T > 0 integer;            # 計画期間数
param p {J} > 0;                # 作業 j の作業時間
param c {J, 1..T} >= 0;         # 作業 j を期 t に開始する費用
param a {J, R, 0..T} >= 0;      # 開始から t 期後の資源 r の使用量
param RUB {R, 1..T} >= 0;       # 期 t の資源 r の上限
set TX = {j in J, t in 1..T-p[j]+1};   # 開始しうる期の組
var x {TX} binary;              # 作業 j を期 t に開始するとき 1
var s {J} >= 0;                 # 作業 j の開始時刻
minimize Cost: sum {(j,t) in TX} c[j,t] * x[j,t];
subject to Assign {j in J}:
   sum {t in 1..T-p[j]+1} x[j,t] = 1;
subject to StartDef {j in J}:
   sum {t in 1..T-p[j]+1} (t-1) * x[j,t] = s[j];
subject to Resource {r in R, t in 1..T}:
   sum {j in J, t2 in max(t-p[j]+1, 1)..min(t, T-p[j]+1)}
      a[j,r,t-t2] * x[j,t2] <= RUB[r,t];
subject to Precedence {(j,k) in PR}:
   s[k] - s[j] >= p[j];
```

お家の例題の最適スケジュール



お家を造るための作業手順と使用する資源量

- 費用 c[j,t] は完了時刻 t−1+p[j]
- 作業員の上限は3日目だけ1人
- 開始日：土台1日目、壁2日目
  - 内装4日目、屋根5日目
- 全作業の完了は6日で最適

実務での注意とモデリングのコツ

欄外：スケジューリング最適化

- 大規模問題には専用ソルバーを推奨
- 制約最適化ソルバーは割当型が得意
  - 時間割、スタッフスケジューリング
- 順序・開始時刻を決める問題は専門ソルバー
  - 例：Pythonから使える OptSeq

問題にあった定式化を行え

- スケジューリングでも様々な定式化がある
- 最も優れた定式化というものはない
- 問題（問題例）により推奨は変わる
- 将来の付加条件も考えて選択する

まとめ

- ✓ 離接定式化は big-M と0-1変数で順序を表す（AMPLでは or で直接書ける）
- ✓ 完了時刻定式化は妥当不等式と分離による切除平面法
- ✓ 時刻添え字定式化は費用 C[j,t] を変えて様々な目的に対応
- ✓ 線形順序付け定式化は推移律の制約で big-M を使わない
- ✓ 位置データ定式化で順列フローショップ問題を表す
- ✓ 資源制約付き問題は時刻添え字で資源量と先行制約を扱う