都市情報システム実習（檀担当分）　[第 2 週]

1　本日の実習テーマ

- 食品製造問題 [1]
- 数独
- （ぶどうの房パズル）

2　食品製造問題

以下の問題を最適化問題として定式化し，glpk を用いて解きましょう：

ある食品は，原料油を精製し，それらをブレンドして製造される．原料油には植物油 VEG 1, VEG 2 と非植物油 OIL 1, OIL 2, OIL 3 がある．各原料油の購入価格は表 1 のように変動する（単位：ドル/トン）．また，最終製品の販売価格は 150 ドル/トン である．

表 1: 各原料油の月別購入価格

	VEG1	VEG2	OIL1	OIL2	OIL3
1 月	110	120	130	110	115
2 月	130	130	110	90	115
3 月	110	140	130	100	95
4 月	120	110	120	120	125
5 月	110	120	150	110	105
6 月	90	100	140	80	135

植物油と非植物油は別の生産ラインで精製される．どの月も，植物油は最大 200 トン，非植物油は最大 250 トンの精製しかできない．精製工程での精製ロスはない．また，精製コストは無視してよい．

各原料油は，翌月以降に使う目的で，それぞれ最大 1,000 トンを在庫として繰り越すことができる．貯蔵費は，植物油・非植物油とも 5 ドル/トン・月である．最終製品は在庫として繰り越せない．精製された原料油も在庫として繰り越せない．なお，1 月が始まる時点で各原料油はそれぞれ 500 トンの在庫があるものとする．また，6 月末には各原料油の在庫がそれぞれ 500 トンでなくてはならないものとする．

最終製品には硬度に関しての技術的制約がある．最終製品の硬度は 3 以上 6 以下でなくてはならない．ブレンドされたものの硬度は加重平均で計算される．原料油の硬度は表 2 の通りである．

この会社が利益を最大にするためには，どのような原料購入計画方針と製造方針を採るべきか？

以下，定式化（のヒント）について説明していきます．

表 2: 各原料油の硬度

VEG1	VEG2	OIL1	OIL2	OIL3
8.8	6.1	2.0	4.2	5.0

まず，いくつかの集合を考える必要があるでしょう．この問題では 1 月から 6 月の計画を立てる必要があるので，月の集合

```
set M := 1 2 3 4 5 6;
```

を準備します（※モデル中では set M; のみでよいことに注意．前回のスライドや，前回作成したモデルファイル，データファイルを参考にすると良い．以下同様）．また，原料の集合を考える必要がありますが，上の問題設定を読むと，原料全体（すなわち植物油と非植物油を合わせた全体）と，植物油・非植物油別に制約を考えることがありそうなので，それぞれの集合を準備することにしましょう．

```
set I := VEG1 VEG2 OIL1 OIL2 OIL3;
set V := VEG1 VEG2;
set O := OIL1 OIL2 OIL3;
```

パラメータとしては，以下のものを準備しておきます:

```
param p {M, I};
param h {I};
```

上から順に「月別・原料別の価格」「原料別硬度」です．また考えるべき変数は以下のようになります：

```
var b{M, I} >= 0;
var r{M, I} >= 0;
var s{M, I} >= 0;
```

上から順に「月別・原料別の購入量」「月別・原料別の精製量」「月別・原料別の在庫量」です．

では次に，制約条件を示していきます．各制約条件が上で説明したどの制約に相当しているか，考えてみてください（個人面談の時に質問します！）:

```
subject to C1 {m in M}:
    sum {v in V} r[m, v] <= 200;

subject to C2 {m in M}:
    sum {o in O} r[m, o] <= 250;

subject to C3 {i in I}:
    s[1, i] == 500 + b[1, i] - r[1, i];

subject to C4 {m in M, i in I: m >= 2}:
```

```
        s[m, i] == s[m - 1, i] + b[m, i] - r[m, i]

    subject to C5 {i in I}:
        s[6, i] == 500;

    subject to C6 {m in M, i in I}:
        s[m, i] <= 1000;

    subject to C7 {m in M}:
        3.0 * sum{i in I} r[m, i]
        <= sum {i in I} (h[i] * r[m, i]);

    subject to C8 {m in M}:
        sum {i in I} (h[i] * r[m, i])
        <= 6.0 * sum{i in I} r[m, i];
```

なお，C4 に現れる {... : m >= 2} とは，m が 2 以上の範囲でのみこの制約を準備する，ということです．

最後に目的関数を作成する必要があります．…が，これは自分で考えてみましょう．目的関数に表れるべきは，

- 販売による収入（なお，最終製品は精製量の合計と一致します）
- 原料購入による支出
- 在庫コストによる支出

になります．

なお，正しく問題が定式化されていれば，目的関数値は 107842.5926 になるはずです．

3 数独

ここでは，最適化ソルバを使って数独を解いてみましょう．一見最適化問題ではないですが，実は解くことができるのです！なお，数独については知っておられる方が多いと思いますが，もしご存じなければ以下を見て下さい：

http://ja.wikipedia.org/wiki/数独

さて，数独には以下のルールがあります：

- 空いているマスに 1～9 のいずれかの数字を入れる．
- 縦・横の各列及び 3×3 のブロック内に同じ数字が複数入ってはいけない．

このルールを最適化問題の制約条件として考え，問題を解くわけです.『目的関数は？』と思う方もいるでしょう．目的関数は…ありません！「制約条件」＝「ルール」を満たす解が得られればそれで良いわけです．

以下では，数独の最適化問題としての定式化を説明します．これに準じたモデル・データを作って問題

まず，次のような集合を考えます．

```
set I := 1 2 3 4 5 6 7 8 9;
set J := 1 2 3 4 5 6 7 8 9;
set K := 1 2 3 4 5 6 7 8 9;
```

I は行，J は列を表し，K は 1 から 9 の数字を表しているものとします．

このとき，次のような変数を考えます：

$$x_{ijk} := \begin{cases} 1, & \text{マス } (i,j) \text{ に数字 } k \text{ が入る} \\ 0, & \text{それ以外} \end{cases}$$

これは 0-1 変数ですから，

```
var x{I, J, K} binary;
```

と宣言すればいいですね．

この変数について，次のような制約を課す必要があります：

$$\begin{aligned} \sum_{i \in I} x_{ijk} &= 1 \quad (j \in J, k \in K) \\ \sum_{j \in J} x_{ijk} &= 1 \quad (i \in I, k \in K) \\ \sum_{k \in K} x_{ijk} &= 1 \quad (i \in I, j \in J) \end{aligned}$$

それぞれの制約がどういう意味を持っているのか，考えた上で glpk 形式で記述してください（どう記述すれば良いのかも考えてみてください）．

さらに，

```
set P := 1 2 3;
set Q := 1 2 3;
set M := 0 1 2;
set N := 0 1 2;
```

を用いて，

```
subject to C_block {p in P, q in Q, k in K}:
  sum {m in M, n in N} x[3*p-m, 3*q-n, k] == 1;
```

という制約を準備する必要もあります（この制約は何を表していますか？）．

また，問題には，数値が最初から与えられているマスがありますよね．この指定については，モデルファイルで

```
set C dimen 3;

subject to C_fixed {(i, j, k) in C}:
    x[i, j, k] == 1;
```

としておき，データファイルで

```
set C :=
    (1, 2, 3), (1, 5, 9), (1, 9, 7),
    (2, 1, 4), (2, 6, 8),
    ＜中略＞
    (9, 1, 5), (9, 5, 4), (9, 8, 9);
```

などとすることで設定することができます（その理由は？）.

最後に目的関数については，

```
minimize Objective:
    0;
```

としましょう（実質，目的関数はない，ということです）.

定式化ができたら，Web 等から見つけてきた適当な問題例に対して，glpk で答えを求めてみてください（問題例は他の人とかぶらないように気をつけて！）. なお，この問題の場合，得られた答が正しいかどうかは，実際に数独のルールを満たしているかどうかを（マス目に埋めて）チェックすればいいですね.

【補足】解ファイル w2p2_result.txt から x の値が 1 の箇所だけ抜き出す方法

コマンドプロンプトで

`grep -e "x.*¥* *1" < w2p2_result.txt` という命令を実行するとよい.

4 Advanced: ぶどうの房パズル

※この章は，余裕のある人・再履修の人など向けです. レポートに含めなくても構いませんが，レポートに含めた場合，加点があり得ます.

4.1 ぶどうの房パズル

「ぶどうの房パズル」というよく知られたパズルがあります. これは，図 1 のように，丸をぶどうの房のように（逆三角形状に）並べ，あるルールに従って丸の中に数字を入れていくというものです. そのルールとは，

ルール 1 使う数字は 1 から「ぶどうの粒の数（例えば図 1 なら 3)」までの正の整数.

ルール 2 同じ数字は 1 度しか用いることができない.

ルール 3 最上段以外では,「自分自身の上にある 2 つの丸に入る数字の差（大きい数字 − 小さい数字)」を入れる.

というものです.

では，房が 2 段の場合を考えてみましょう. 図 2, 3 は，上記のルール 1 〜 3 を守っているので，ぶどうの房パズルの正解になります（正解は一通りとは限りません. また，各図の鏡像（左右を入れ替えたもの）も正解です）. しかし，図 4 は，ルール 3 を守っていないため不正解です.

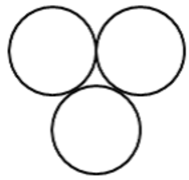

図 1: ぶどうの房パズル（房が 2 段の場合）

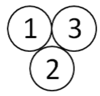

図 2: ぶどうの房パズル・正解例 1

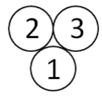

図 3: ぶどうの房パズル・正解例 2

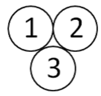

図 4: ぶどうの房パズル・不正解例

4.2 最適化問題としての定式化

では，ぶどうの房パズルを最適化問題として定式化してみましょう. 一例として，次のような定式化が可能です：

```
set I;
set J;
set K dimen 3;

param M;
var   x{I};
var   d{I, J} binary;
var   t{I} binary;

minimize Objective: 0;

subject to C1 {i in I}:
x[i] == sum{j in J} (j * d[i, j]);

subject to C2 {i in I}:
sum{j in J} d[i, j] == 1;

subject to C3 {j in J}:
sum{i in I} d[i, j] == 1;

subject to C4_L {(p, q, r) in K}:
- M * t[r] + (x[q] - x[p]) <= x[r];

subject to C4_U {(p, q, r) in K}:
x[r] <= M * t[r] + (x[q] - x[p]);

subject to C5_L {(p, q, r) in K}:
- M * (1 - t[r]) + (x[p] - x[q]) <= x[r];
```

```
subject to C5_U {(p, q, r) in K}:
x[r] <= M * (1 - t[r]) + (x[p] - x[q]);
```

集合 I, J は 1 から「ぶどうの粒の数」までの正の整数の集合です．例えばぶどうの房が 3 段の場合は，次のようになります：

```
set I := 1 2 3 4 5 6;
set J := 1 2 3 4 5 6;
```

また集合 K は，ルール 3 を考える粒 3 つの逆三角形のまとまりを集めたものです．例えばぶどうの房が 3 段の場合，図 5 のようにぶどうの粒に番号を付け，集合 K を

```
set K := (1, 2, 4), (2, 3, 5), (4, 5, 6);
```

と定義します．

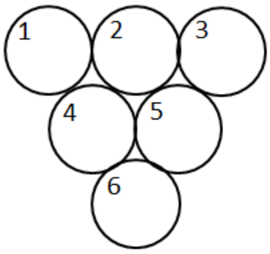

図 5: ぶどうの粒への番号付け

変数 x[i] は，粒 i に入る数字を表しています（整数値になりますが，連続変数として定義すれば十分です）．また変数 d[i, j] は，粒 i に入る数字が j であれば 1，そうでなければ 0 になるような 0-1 変数です．

制約 C1，C2，C3 はルール 1,2 に対応しています：C1 は，粒 i に入る数字を定めている制約式です．右辺は，d[i, j] が 1 のところだけが積算されるわけですが，C2 により，各 i について d[i, j] が 1 になるような j は 1 つしかないことになります．さらに C3 は，各数字 j が 1 回しか使えないことに対応しています．

さらに，制約 C4_L，C4_R，C5_L，C5_R はルール 3 に対応しています：パラメータ M は十分大きな正の整数（例えば粒の数の 2 倍にしておけば十分[1]）です．また変数 t[r] は 0-1 変数です．その意味を考えてみるため，t[r] が 0 のときと 1 のときで場合分けをしてみましょう．

t[r] = 0 のとき，制約 C4_L，C4_R をまとめると次のようになります：

```
x[q] - x[p] <= x[r] <= x[q] - x[p]
```

これは，x[r] == x[q] - x[p] であることを意味します．一方，制約 C5_L，C5_R をまとめると，

```
- M + (x[p] - x[q]) <= x[r] <= M + (x[p] - x[q])
```

となります．M は十分大きな正の整数ですから，この制約式は事実上意味のない制約（最左辺が十分小さく，最右辺が十分大きくなる）になります．

一方，t[r] = 1 のとき，C4_L，C4_R，C5_L，C5_R は次のようになります：

```
- M + (x[q] - x[p]) <= x[r] <= M + (x[q] - x[p])
x[p] - x[q] <= x[r] <= x[p] - x[q]
```

すなわち x[r] == x[p] - x[q] ということになります．

これらのことより，t[r] は，粒 r の上にある粒 p，q に入る数字の大小関係を表しており，

- t[r] = 0 のとき：x[p] < x[q] であり，
  x[r] == x[q] - x[p]
- t[r] = 1 のとき：x[p] > x[q] であり，
  x[r] == x[p] - x[q]

ということになります．

最後に目的関数ですが，特に設けなくてもよいので，ここでは 0 としています．あるいは，複数ある答の中で何か特徴のあるもの（例えば一番左上の粒に入る数字が最大/最小になるもの）を求めたいのであれば，それを目的関数としても構いません．

これにより，ぶどうの房パズルを最適化問題として定式化できたことになります．

4.3 問題

次のパターン（図 6 〜 8）で答えを探してみましょう．また，これ以上段数を増やしたらどうなるでしょうか？

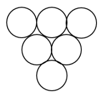

図 6: ぶどうの房パズル（3 段）

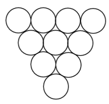

図 7: ぶどうの房パズル（4 段）

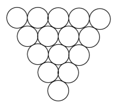

図 8: ぶどうの房パズル（5 段）

参考文献

[1] H. P. Williams, Model Building in Mathematical Programming (Fourth ed.), John Wiley & Sons (1999).
前田英次郎監訳，小林英三訳，数理計画モデルの作成法（産業図書）．

[1] なぜ粒の数の 2 倍で十分なのか，考えてみましょう．