数理最適化　レポート課題1

パズルの定式化とそれに関する計算機実験

提出締切　2020年8月7日

情報学群情報科学類　3年
学籍番号：201811319
氏名：永崎遼太

目次

1. 実験の概要

1.1 目標

　様々な問題に対してその問題定式化の方法を学び、最適化ソルバーを用いることによってそれらの問題を解くことができるようになる。

1.2 実験の内容

　以下の3つのことを行う。

- 最適化ソルバー（Python + Gurobi / CPLEX etc.）を使って整数計画問題を解く。（cf. テキスト [1], [2]）
- 様々な定式化手法を習得し、解きたい問題を適切な数理モデルに定式化できるようにする。（cf. テキスト [2]）
- パズルを整数計画問題として定式化し、GurobiやCPLEXなどの整数計画ソルバーを用いて計算機実験を行う。本レポートではこの実験内容についてまとめる。

1.3 テキスト

　以下の2つのテキストを参考として実験を行う。

[1] 「 データ分析ライブラリーを用いた最適化モデルの作り方 」 斉藤努（著） （近代科学社） 2018年

[2] 「 あたらしい数理最適化　Python言語とGurobiで解く 」 久保幹雄・J.P.ペドロソ・村松正和・A.レイス（著） （近代科学社） 2012年

2. 実験内容

2.1 パズルの紹介

　本実験では以下のパズルについて定式化し、それに関する計算機実験を行った。

1. ノノグラム
2. カックロ（クロスサム）

2.2 実験方法

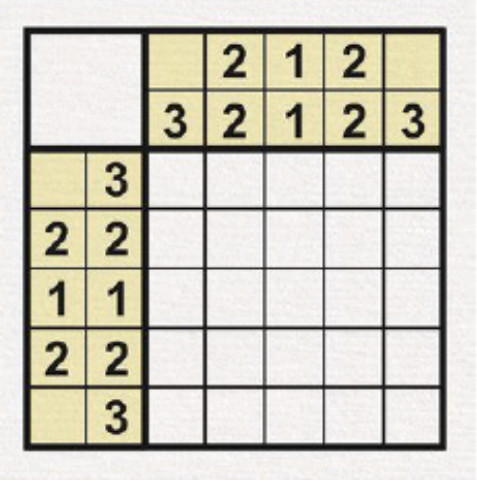
ルールに基づいて、制約条件、目的関数を数式で表現し、パズルを整数最適化問題へ定式化する。そして、Python+Gurobi/CPLEXを用いて計算機実験を行う。その際、以下のことを確認する。

- 正しい解を出力するか
- 平均計算時間はどれだけか
- 他の定式化方法がないか
- 問題サイズの大小によって結果がどのようになるか

3. ノノグラム

3.1 ノノグラムのルール

ノノグラムは、図のような任意のサイズの正方形盤面がはじめに与えられる。図はサイズが5×5である問題の例である。



問題例

各行の左、各列の上にある数字は、その行・列の中で連続して黒く塗る白マスの数を表している。1つの行・列に対して数字が複数ある場合は、数字の並び順通りにその数字の数だけ連続して黒く塗る。また、1つの行・列に対して数字が複数ある場合、その数字が表す黒鱒の連続の間に1マス以上の白マス（塗らないマス）が入る。
このルールに基づくと、上図における特定の行における解候補は次のようになる。

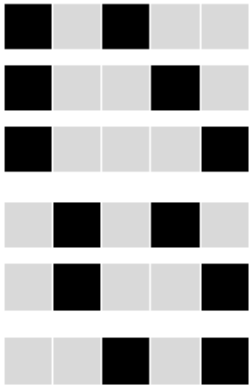
一行目

解候補

二行目

解候補

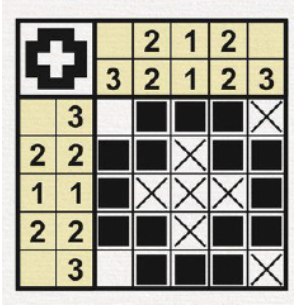
三行目



解候補

ノノグラムは、与えられた盤面の白いマスに対して、各行・列に対して、その間に1マス以上の間隔を取りながら、ヒントで与えられたマスの分を連続して黒く塗る。その結果、目標とする絵が浮かび上がることが確認できれば、パズルを解けたと言える。

例としてあげた上の問題では、次のような絵が浮かび上がれば、パズルが解けたことになる。



問題例の答え

3.2 ノノグラムの定式化

5×5のサイズの問題を整数計画問題として定式化する。

3.2.1 変数の設定

行 $i = 1,...,5$ 、列 $j = 1,...,5$ が指定するマスの目の色を次の変数 $v_{i,j}$ で表す。

$$v_{i,j} \in \{0, 1\}$$

$v_{i,j}$ が0である時は、上からi番目、左からj番目のマスは塗りつぶさず白のままで、$v_{i,j}$ が1である時は、上からi番目、左からj番目のマスは塗りつぶし黒となる。

3.2.2 制約条件の設定

各行・列における全ての解候補から1つを採用する。次の変数 r_k を用いる。

$$r_k \in \{0, 1\},\ \forall k$$

r_k が0である時は、k番目の解候補を不採用とし、r_k が1である時は、k番目の解候補を採用する。

3.2.3 目的関数の設定

パズルは実行可能解を求める問題であり、目的関数を最適化する問題ではない。そのため、目的関数はなんでも良い。

以上の定式化をまとめると、次のようになる。

変数	$v_{ij} \in \{0,1\}\ \forall i,j$　マスi,jが黒かどうか
	$r_k \in \{0,1\}\ \forall k$, 縦または横　縦または横ごとに$k$番目の候補を選ぶかどうか
subject to	$\sum_k r_k = 1\ \forall$縦または横　縦または横ごとに候補の中から1つ
	候補を選んだらマスの色は候補の通り

3.3 ソルバーの実装

3.3.1 実装の方針

以下の方針で、Pythonによりソルバーを実装する。

1. ライブラリのインポート
2. 入力(与えられているヒント)の定義
3. 定式化
    1. 解候補を列挙
    2. 制約条件の定義
    3. 目的関数の定義
4. 最適化の実行
5. 結果の表示

3.3.2 実装コードおよびコードの説明

Jupyter Notebook にて、nonogram_solve.ipynb ファイルを作成した。

作成したコードについて、3.3.1節で説明した実装の方針に基づいてコードを説明する。

---

1. ライブラリのインポート

```
#!pip3 install pulp ortoolpy
%matplotlib inline
import numpy as np, matplotlib.pyplot as plt
from pulp import LpProblem, lpSum, value
from ortoolpy import addvars, addbinvars
```

ライブラリをインポートしている。

2. 入力(与えられているヒント)の定義

```
# 5×5
# 行に対してのヒント
row = [[int(s) for s in t.split(',')] for t in
        '3 2,2 1,1 2,2 3'.split()]
# 列に対してのヒント
column = [[int(s) for s in t.split(',')] for t in
          '3 2,2 1,1 2,2 3'.split()]
```

行・列に対してのヒントを定義している。

行のヒントを出力すると、次のようになる。

```
In [7]: row
```

```
Out[7]: [[3], [2, 2], [1, 1], [2, 2], [3]]
```

列のヒントを出力すると、次のようになる。

```
In [8]: column
```

```
Out[8]: [[3], [2, 2], [1, 1], [2, 2], [3]]
```

3. 定式化

3.1 解候補を列挙

解候補を列挙する関数を次のように定義する。

```
def makelist(i, j, k):
    return [0] * i + [1] * j + [0] * k # 0がi個、1がj個、0がk個並んだ配列を返す
# ex) [0]*4+[1]*3+[0]*2 → [0, 0, 0, 0, 1, 1, 1, 0, 0]
```

この関数を実行すると、次のようになる。

```
In [11]: n=6
         p=2
         [makelist(i, p, n - p - i) for i in range(n - p + 1)]
```

```
Out[11]: [[1, 1, 0, 0, 0, 0],
          [0, 1, 1, 0, 0, 0],
          [0, 0, 1, 1, 0, 0],
          [0, 0, 0, 1, 1, 0],
          [0, 0, 0, 0, 1, 1]]
```

この関数を利用して、各行・列における解候補を列挙したリストを作成する関数を定義する。

```
def sol_list(n, l):
    p = l[-1]
    if len(l) == 1:
        if n < p: return None
        return [makelist(i, p, n - p - i) for i in range(n - p + 1)]
    ll = l[:-1]
    s = sum(ll) + len(ll) - 1
    return [j + makelist(1, p, n - p - s - i - 1) \
        for i in range(n - p - s) for j in sol_list(i + s, ll)]
```

この関数を実行すると次のようになる。

```
In [13]: for i, hh in enumerate(column):
             l = sol_list(len(column), hh) # (vの列数, ヒントの内容)
             print(l)
             print('--')
         # column: [[3], [2, 2], [1, 1], [2, 2], [3]]
```

```
[[1, 1, 1, 0, 0], [0, 1, 1, 1, 0], [0, 0, 1, 1, 1]]
--
[[1, 1, 0, 1, 1]]
--
[[1, 0, 1, 0, 0], [1, 0, 0, 1, 0], [0, 1, 0, 1, 0], [1, 0, 0, 0, 1], [0, 1, 0, 0, 1], [0, 0,
1, 0, 1]]
--
[[1, 1, 0, 1, 1]]
--
[[1, 1, 1, 0, 0], [0, 1, 1, 1, 0], [0, 0, 1, 1, 1]]
--
```

3.2 / 3.3 制約条件・目的関数の定義

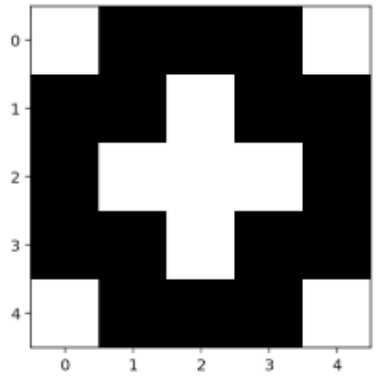
```
def do(m, v, hint):
    for i, hh in enumerate(hint):
        l = sol_list(v.shape[0], hh) # 引数：(問題のサイズ, ヒントの内容)→各行(列)に対する解候補を列挙したリストを作成
        r = addbinvars(len(l)) # rk∈{0,1} ∀k,行または列　行(列)ごとに解候補を列挙したリスト内のk番目の候補を選ぶかどうか　1→選ぶ
        m += lpSum(r) == 1 # ∑krk=1 ∀行または列　行または列ごとに候補の中から1つ選ぶ
        for j, c in enumerate(l):
            for k, b in enumerate(c):
                m += (1 - 2 * b) * v[k,i] <= 1 - b - r[j] # 候補を選んだ通りにマスの色を塗る
```

4. 最適化の実行

```
In [16]: m = LpProblem()
         v = np.array(addvars(len(row), len(column))) # vij∈{0,1} ∀i,j    マスi,jが黒かどうか
         do(m, v, row) # 列について
         do(m, v.T, column) # 行について
         # %timeit -n5 m.solve()
         %time m.solve()
         # m.solve()

         CPU times: user 2.79 ms, sys: 4.3 ms, total: 7.09 ms
         Wall time: 35.7 ms

Out[16]: 1
```

5. 結果の表示

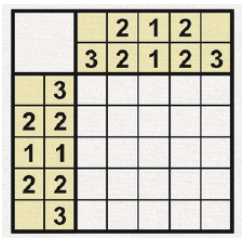
```
%config InlineBackend.figure_formats={'png','retina'}
plt.imshow(1-np.vectorize(value)(v), cmap='gray', interpolation='none');
```



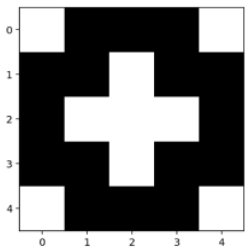
実際に作成したコードは5節に掲載している。

3.4 実行結果

左図のような入力を与えた結果、右図の結果が得られた。



入力



出力

この時、入力は次のようにして表される。

Row:
[[3], [2, 2], [1, 1], [2, 2], [3]]

Column:
[[3], [2, 2], [1, 1], [2, 2], [3]]

目的とする絵が浮かび上がったことが確認できた。
また、この時の計算時間は次のようになった。
17.5 ms ± 2.36 ms per loop (mean ±} std. dev. of 7 runs, 5 loops each)

さらに、入力のサイズを変えて実行した結果は次のようになる。

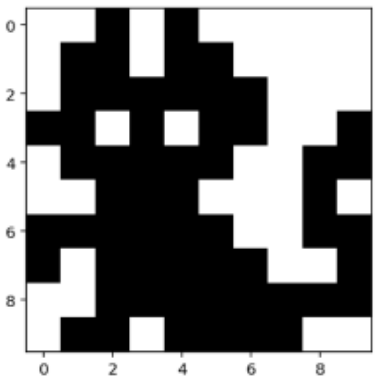
・問題サイズが10×10である場合

[入力]

Row:
[[1, 1], [2, 2], [5], [2, 1, 2, 1], [5, 2], [3, 1], [6, 2], [1, 5, 1], [8], [2, 4]]

Column:
[[1, 2], [4, 1, 1], [3, 6], [7], [3, 6], [4, 4], [1, 3], [2], [3, 1], [2, 3]]

[出力]



[実行時間]

CPU times: user 41.2 ms, sys: 5.24 ms, total: 46.4 ms
Wall time: 93 ms

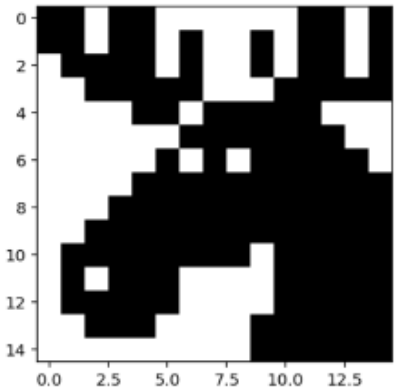
・問題サイズが15×15である場合

[入力]

Row:
[[2, 2, 2, 1], [2, 2, 1, 1, 2, 1], [4, 1, 1, 2, 1], [5, 5], [2, 5], [7], [1, 1, 5], [11], [12],[13], [8, 5], [1, 3, 5], [5, 5], [3, 6], [6]]

Column:
[[2], [3, 3], [2, 2, 2], [4, 6], [5, 7], [2, 7], [3, 1, 4], [7], [2, 4], [2, 6, 2], [12], [15], [4, 10], [1, 9], [4, 8]]

[出力]



[実行時間]

CPU times: user 251 ms, sys: 15.4 ms, total: 266 ms
Wall time: 1.14 s

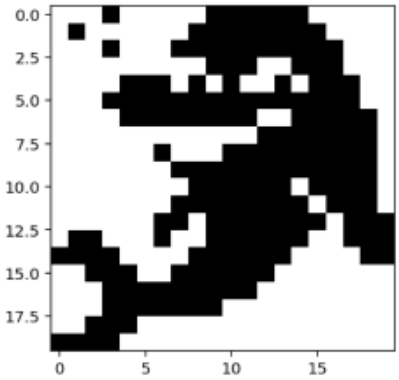
・問題サイズが20×20である場合

[入力]

Row:
[[1, 6], [1, 8], [1, 10], [3, 3], [3, 1, 1, 1, 3], [15], [8, 5], [7], [1, 9], [12], [6, 4], [8, 3], [2, 7, 3], [2, 1, 6, 3], [4, 6, 2], [3, 6], [9], [7], [3], [4]]

Column:
[[1, 1], [1, 2, 1], [3, 2], [1, 1, 1, 6], [3, 4], [3, 2], [3, 1, 2, 2], [1, 2, 1, 2, 3], [2, 3, 3, 4], [4, 2, 9], [7, 9], [4, 2, 9], [3, 1, 9], [3, 2, 8], [4, 5, 3], [10, 1], [10], [10], [9], [3]]

[出力]



[実行時間]

CPU times: user 2.78 s, sys: 97.2 ms, total: 2.88 s
Wall time: 1min 30s

以上の実験は、結果からも分かるとおり、目的とする絵が浮かび上がり、パズルを正しく解くことができていると言える。
解が求まらないようなヒントが与えられている次の場合について実験を行うと、実行可能領域が空となり、結果は次のようになった。

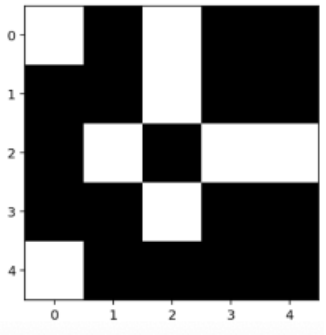
・問題サイズが5×5である場合

[入力]

Row: [[3], [2, 2], [1, 1], [2, 2], [2]]

Column: [[2], [2, 2], [1, 1], [2, 2], [3]]

[出力]



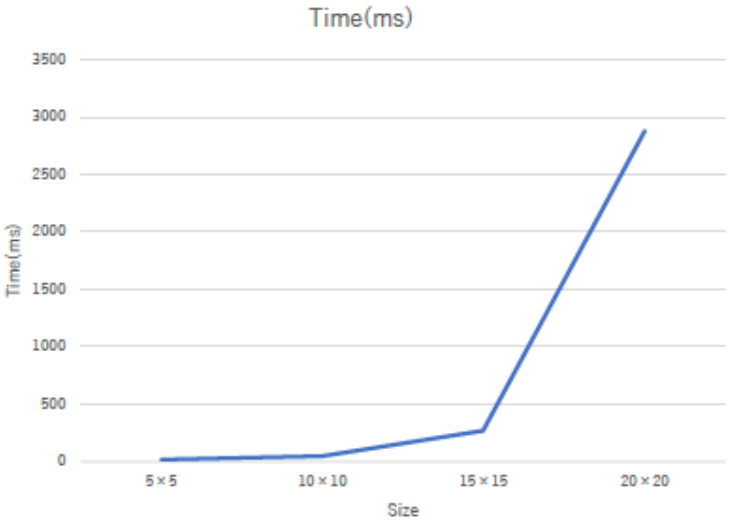
[実行時間]

CPU times: user 6.31 ms, sys: 6.12 ms, total: 12.4 ms
Wall time: 19.9 ms

3.5 考察

　入力サイズを大きくしても、解が存在する問題について、ソルバーを実行することでパズルを解くことができた。そのため、モデリングを正しく行うことができていると確認できる。

問題サイズごとの、実行時間をグラフにすると、次のようになる。



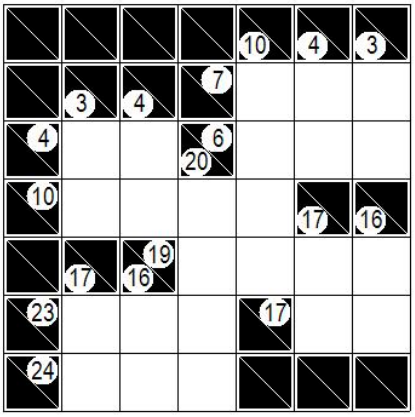
問題サイズごとの実行時間

問題サイズが大きくなると、実行時間が急増することがわかる。これは、現在のモデリングでは、全ての行・列に対する解候補を等しく扱い、最適解を探索しているためであると考えられる。そのため、解候補が少ない行・列から優先的に解を探索するようにするという改善方法が考えられる。

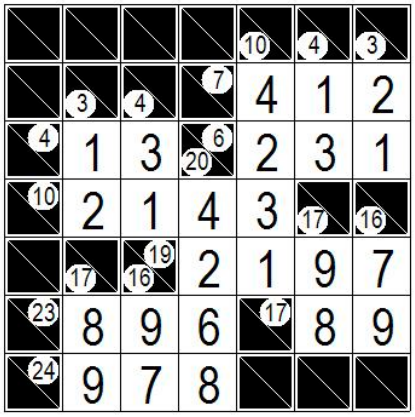
4. カックロ（クロスサム）

4.1 カックロのルール

カックロは、下図のような任意のサイズの正方形盤面がはじめに与えられる。



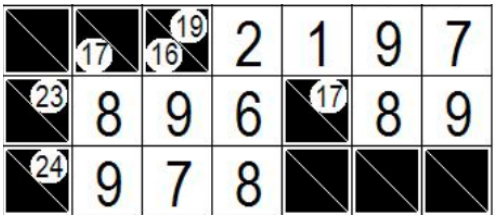
問題例



問題例の答え

カックロは、シロマスの中に1〜9の数字を入れる。与えられた盤面の斜めに仕切られたマスのある数字のうち、右上にある数字が横へ続く空マスの合計、左下にある数字が縦に続く空マスの合計を表している。合計の値がヒントの値となるように空マスに数字を入れる。この時、横・縦に連続する空マスの中に同じ数字は入らない。

実際に、例で挙げた問題と解くと、上の右図のようになる。一部を取り出して見ていると、確かに横・縦のマスの合計の値がヒントの通りになっていることが確認できる。



縦マス、横マスの合計

4.2 カックロの定式化

4.2.1 変数の設定

縦の合計に対するヒントを格納する変数を$hint_v$、横の合計に対するヒントを格納する変数を$hint_h$で表す。それぞれの変数では、開始行、開始列、連続する空マス、合計の値を格納する。

i 行、j 列が指定するマスの数字がk+1であるかを $v_{i,j,k}$ で表す。

$$v_{i,j,k} \in \{0,\ 1\},\ \forall i, j, k$$

$v_{i,j,k}$が0である時は、上からi番目、左からj番目のマスの数字はk+1ではなく、$v_{i,j,k}$ が1である時は、上からi番目、左からj番目のマスの数字はk+1となる。

i 行、j 列が指定するマスの数字を $r_{i,j}$ で表す。

$$r_{i,j} \in Z\ ,\ \forall i, j$$

$r_{i,j}$ に入る値は整数全体の要素である。

4.2.2 制約条件の設定

i 行、j 列が指定するマスには数字1つしか入らない。

$$\sum_k v_{ijk} = 1\ \forall i, j$$

変数rをvで表現する。

$$\sum_k k \times v_{ijk} + 1 = r_{ij}\ \forall i, j$$

マスの合計が同じになる。

$$\sum_{ij} r_{ij} = sum \ \forall i, j$$

4.2.3 目的関数の設定

パズルは実行可能解を求める問題であり、目的関数を最適化する問題ではない。そのため、目的関数はなんでも良い。

以上の定式化をまとめると、次のようになる。

$$
\begin{aligned}
\text{変数} \quad & v_{ijk} \in \{0,1\} \ \forall i,j,k \quad \text{マス} i,j \text{が数字} k+1 \text{か} \\
& r_{ij} \in Z \ \forall i,j \quad \text{マス} i,j \text{の数字} \\
\text{subject to} \quad & \textstyle\sum_k v_{ijk} = 1 \ \forall i,j \quad \text{数字は1つ} \\
& \textstyle\sum_k k \times v_{ijk} + 1 = r_{ij} \ \forall i,j \quad r \text{を} v \text{で表現} \\
& \textstyle\sum_{ij} r_{ij} = sum \ \forall i,j \quad \text{マスの合計が同じ}
\end{aligned}
$$

4.3 ソルバーの実装

4.3.1 実装の方針

以下の方針で、Pythonによりソルバーを実装する。

1. ライブラリのインポート
2. 入力(与えられているヒント)の定義
3. 定式化
4. 最適化の実行
5. 結果の表示

4.3.2 実装コードおよびコードの説明

Jupyter Notebook にて、kakkuro_solve.ipynb ファイルを作成した。

作成したコードについて、4.3.1節で説明した実装の方針に基づいてコードを説明する。

---

1. ライブラリのインポート

```
#!pip3 install pulp ortoolpy
import re, numpy as np, pandas as pd
from pulp import LpProblem, lpDot, lpSum, value
from ortoolpy import addvar, addvars, addbinvar, addbinvars
```

ライブラリをインポートしている。

---

2. 入力(与えられているヒント)の定義

```
data = """\
###...
..#...
....##
##....
...#..
...###""".splitlines()
```

この入力データを元に、空きマスのデータをdataとして定義している。コードでは、空きマスは ” . “、空きマスでないマスは “ # “ として定義している。

ヒントを表す変数$hint_v$ 、$hint_h$ をそれぞれ hint_v, hint_h として次のように定義し、それぞれ開始行、開始列、空きマスの個数、合計のデータを格納している。

```
hint_v = [ # 開始行、開始列、個数、合計
    (0,3,4,10),
    (0,4,2,4),
    (0,5,2,3),

    (1,0,2,3),
    (1,1,2,4),

    (2,2,4,20),

    (3,4,2,17),
    (3,5,2,16),

    (4,0,2,17),
    (4,1,2,16),
]
hint_h = [ # 開始行、開始列、個数、合計
    (0,3,3,7),

    (1,0,2,4),
    (1,3,3,6),

    (2,0,4,10),

    (3,2,4,19),

    (4,0,3,23),
    (4,4,2,17),

    (5,0,3,24),
]
```

---

3. 解を求める / 4. 定式化

```
ni,nj = len(data),len(data[0])
a = pd.DataFrame([(i,j,k) for i in range(ni) for j in range(nj)
    if data[i][j]=='.' for k in range(1,10)], columns=list('行列数'))
a['Var'] = addbinvars(len(a))
a[:2]
```

ni, nj, aを出力すると次のようになる。

In [39]:
```
print('ni: ',ni)
print('nj: ',nj)
```

```
ni:  6
nj:  6
```

In [40]:
```
print(a)
```

```
     行  列  数      Var
0    0  3  1  v001153
1    0  3  2  v001154
2    0  3  3  v001155
3    0  3  4  v001156
4    0  3  5  v001157
..  .. .. ..      ...
211  5  2  5  v001364
212  5  2  6  v001365
213  5  2  7  v001366
214  5  2  8  v001367
215  5  2  9  v001368

[216 rows x 4 columns]
```

以下で、制約条件、目的関数を定義している。

```
m = LpProblem()
for _,v in a.groupby(['行','列']):
    m += lpSum(v.Var) == 1 # (1)
for (di,dj),hint in zip([(1,0),(0,1)],[hint_v,hint_h]):
    for si,sj,nl,sm in hint:
        b = a.query(f'{si}<=行<={si+nl*di}&{sj}<=列<={sj+nl*dj}')[['数','Var']]
        for _,v in b.groupby('数'):
            m += lpSum(v.Var) <= 1 # (2)
        m += lpDot(*b.T.values) == sm # (3)
%time m.solve()
```

最適化の実行で、実行時間を計測している。

5. 結果の表示

```
a['Val'] = a.Var.apply(value)
r = a[a.Val>0.5].数[::-1].tolist()
print(re.sub('\\.', lambda _: str(r.pop()), '\n'.join(data)))
```

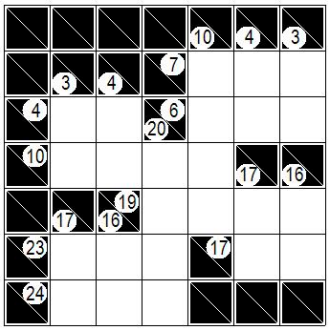
```
###412
13#231
2143##
##2197
896#89
978###
```

2の入力の定義で与えたdataの空きコマの部分に1～9の数字が入ったデータが格納されたデータを返している。

実際に作成したコード5節に掲載している。

4.4 実行結果

左図のような入力を与えた結果、右図の結果が得られた。

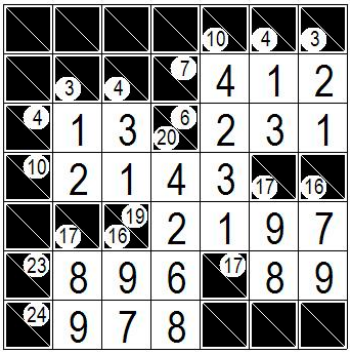


入力

```
###412
13#231
2143##
##2197
896#89
978###
```

出力

実際の入力に対する正解は次のようである。



正解

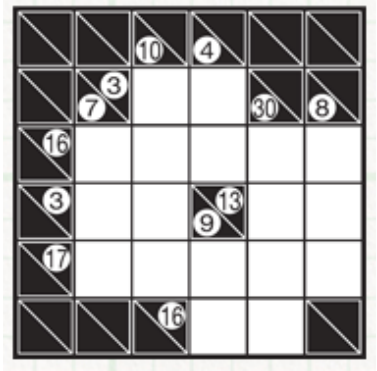
出力結果が正解を正しく表していることが確認できた。
また、この時の計算時間は次のようになった。

```
CPU times: user 4.89 ms, sys: 3.66 ms, total: 8.55 ms
Wall time: 25.1 ms
```

さらに、入力のサイズを変えて実行した結果は次のようになる。

問題サイズが6×6である場合

[入力]

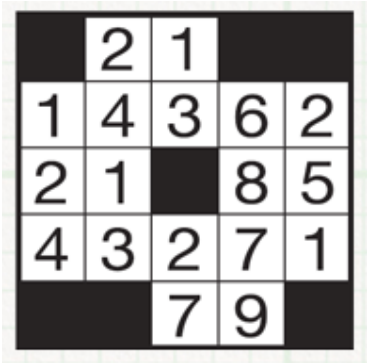


実際の問題

```
data = """\
#..##
.....
..#..
.....
##..#""".splitlines()
hint_v = [ # 開始行、開始列、個数、合計
    (0,1,4,10),
    (0,2,2,4),

    (1,0,3,7),
    (1,3,4,30),
    (1,4,3,8),

    (3,2,2,9),
]
hint_h = [ # 開始行、開始列、個数、合計
    (0,1,2,3),

    (1,0,5,16),

    (2,0,2,3),
    (2,3,2,13),

    (3,0,5,17),

    (4,2,2,16),


]
```

問題の定義

[正解/出力]



正解

```
#21##
14362
21#85
43271
##79#
```

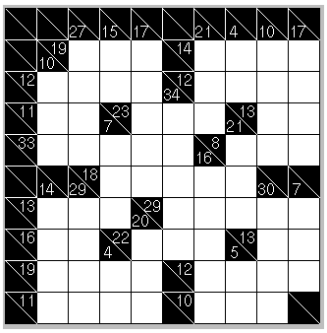
出力

[実行時間]

CPU times: user 3.61 ms, sys: 3.2 ms, total: 6.81 ms
Wall time: 16.4 ms

問題サイズが10×10である場合

[入力]



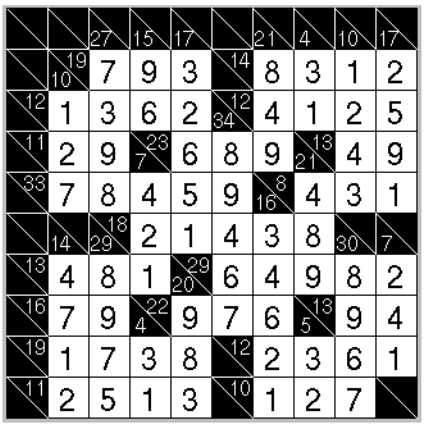
実際の問題

```
data = """\
#...#....
....#....
..#...#..
.....#...
##.....##
...#.....
..#...#..
....#....
....#...#""".splitlines()
```

問題の定義

```
hint_v = [ # 開始行、開始列、個数、合計
    (0,1,4,27),
    (0,2,2,15),
    (0,3,5,17),
    (0,5,3,21),
    (0,6,2,4),
    (0,7,4,10),
    (0,8,4,17),

    (1,0,3,10),

    (2,4,5,34),

    (3,2,3,7),
    (3,6,3,21),

    (4,5,5,16),

    (5,0,4,14),
    (5,1,4,29),
    (5,7,4,30),
    (5,8,3,7),

    (6,3,3,20),

    (7,2,2,4),
    (7,6,2,5),
]
hint_h = [ # 開始行、開始列、個数、合計
    (0,1,3,19),
    (0,5,4,14),

    (1,0,4,12),
    (1,5,4,12),

    (2,0,2,11),
    (2,3,3,23),
    (2,7,2,13),

    (3,0,5,33),
    (3,6,3,8),

    (4,2,5,18),

    (5,0,3,13),
    (5,4,5,29),

    (6,0,2,16),
    (6,3,3,22),
    (6,7,2,13),

    (7,0,4,19),
    (7,5,4,12),

    (8,0,4,11),
    (8,5,3,10),
]
```

[正解/出力]



正解

```
#793#8312
1362#4125
29#689#49
78459#431
##21438##
481#64982
79#976#94
1738#2361
2513#127#
```

出力

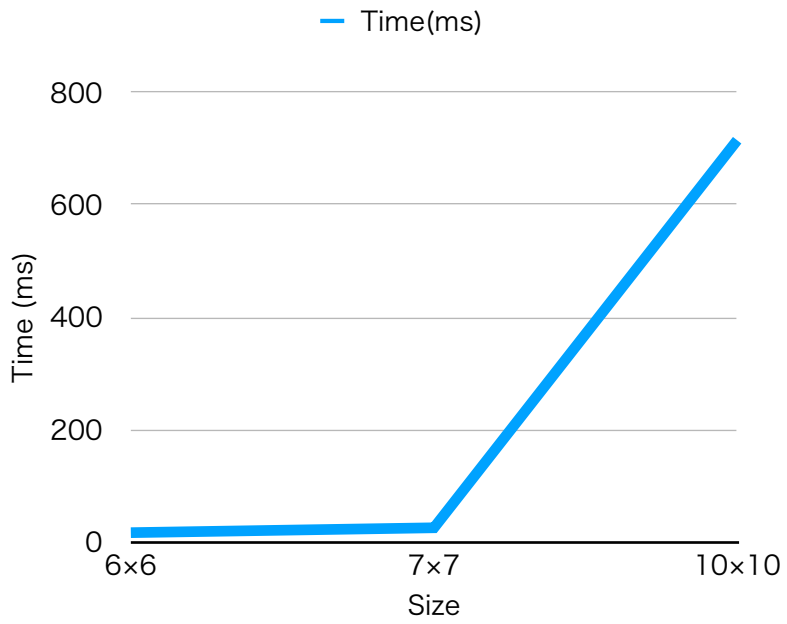
[実行時間]

CPU times: user 12.6 ms, sys: 3.84 ms, total: 16.4 ms
Wall time: 715 ms

4.5 考察

　入力サイズを大きくしても、解が存在する問題について、ソルバーを実行することでパズルを解くことができた。そのため、モデリングを正しく行うことができていると確認できる。

問題サイズごとの、実行時間をグラフにすると、次のようになる。



問題サイズが大きくなると、実行時間が増加していくことがわかる。現在のモデリングに制約条件を加えることで、より高速にパズルと解くことができると考えられるので、今後改善していこうと思う。

5. コード

nonogram_solve.ipynb

1. ライブラリのインポート

```
#!pip3 install pulp ortoolpy
%matplotlib inline
import numpy as np, matplotlib.pyplot as plt
from pulp import LpProblem, lpSum, value
from ortoolpy import addvars, addbinvars
```

2. 入力(与えられているヒント)の定義

```
# 5×5
# 行に対してのヒント
row = [[int(s) for s in t.split(',')] for t in
          '3 2,2 1,1 2,2 3'.split()]
# 列に対してのヒント
column = [[int(s) for s in t.split(',')] for t in
          '3 2,2 1,1 2,2 3'.split()]
```

3. 解を求める

3.1 解候補を作成する関数

```
def makelist(i, j, k):
    return [0] * i + [1] * j + [0] * k # 0がi個、1がj個、0がk個並んだ配列を返す
# ex) [0]*4+[1]*3+[0]*2 → [0, 0, 0, 0, 1, 1, 1, 0, 0]
```

3.2 各行(列)における解候補を列挙したリストを作成する

```
def sol_list(n, l):
    p = l[-1]
    if len(l) == 1:
        if n < p: return None
        return [makelist(i, p, n - p - i) for i in range(n - p + 1)]
    ll = l[:-1]
    s = sum(ll) + len(ll) - 1
    return [j + makelist(1, p, n - p - s - i - 1) \
        for i in range(n - p - s) for j in sol_list(i + s, ll)]
```

3.3 制約条件、目的関数の定義

```
def do(m, v, hint):
    for i, hh in enumerate(hint):
        l = sol_list(v.shape[0], hh) # 引数：(問題のサイズ, ヒントの内容)→各行(列)に対する解候補を列
        r = addbinvars(len(l)) # rk∈{0,1} ∀k,行または列　　行(列)ごとに解候補を列挙したリスト内のk番目
        m += lpSum(r) == 1 # ∑krk=1 ∀行または列　　行または列ごとに候補の中から1つ選ぶ
        for j, c in enumerate(l):
            for k, b in enumerate(c):
                m += (1 - 2 * b) * v[k,i] <= 1 - b - r[j] # 候補を選んだ通りにマスの色を塗る
```

4. 最適化の実行

```
m = LpProblem()
v = np.array(addvars(len(row), len(column))) # vij∈{0,1} ∀i,j　　マスi,jが黒かどうか
do(m, v, row) # 列について
do(m, v.T, column) # 行について
# %timeit -n5 m.solve()
%time m.solve()
# m.solve()
```

```
CPU times: user 2.79 ms, sys: 4.3 ms, total: 7.09 ms
Wall time: 35.7 ms
```

```
1
```

結果の表示

```
%config InlineBackend.figure_formats={'png','retina'}
plt.imshow(1-np.vectorize(value)(v), cmap='gray', interpolation='none');
```

```
#!pip3 install pulp ortoolpy
import re, numpy as np, pandas as pd
from pulp import LpProblem, lpDot, lpSum, value
from ortoolpy import addvar, addvars, addbinvar, addbinvars
```

```
data = """\
###...
..#...
....##
##....
...#..
...###""".splitlines()

hint_v = [ # 開始行、開始列、個数、合計
    (0,3,4,10),
    (0,4,2,4),
    (0,5,2,3),

    (1,0,2,3),
    (1,1,2,4),

    (2,2,4,20),

    (3,4,2,17),
    (3,5,2,16),


    (4,0,2,17),
    (4,1,2,16),

]
hint_h = [ # 開始行、開始列、個数、合計
    (0,3,3,7),

    (1,0,2,4),
    (1,3,3,6),

    (2,0,4,10),

    (3,2,4,19),

    (4,0,3,23),
    (4,4,2,17),

    (5,0,3,24),
]
```

```
ni,nj = len(data),len(data[0])
a = pd.DataFrame([(i,j,k) for i in range(ni) for j in range(nj)
    if data[i][j]=='.' for k in range(1,10)], columns=list('行列数'))
a['Var'] = addbinvars(len(a))
```

```
m = LpProblem()
for _,v in a.groupby(['行','列']):
    m += lpSum(v.Var) == 1
for (di,dj),hint in zip([(1,0),(0,1)],[hint_v,hint_h]):
    for si,sj,nl,sm in hint:
        b = a.query(f'{si}<=行<={si+nl*di}&{sj}<=列<={sj+nl*dj}')[['数','Var']]
        for _,v in b.groupby('数'):
            m += lpSum(v.Var) <= 1
        m += lpDot(*b.T.values) == sm
%time m.solve()
```

...

```
a['Val'] = a.Var.apply(value)
r = a[a.Val>0.5].数[::-1].tolist()
print(re.sub('\\.', lambda _: str(r.pop()), '\n'.join(data)))
```