BL 法による長方形詰込み問題の Python での実装と Grasshopper を用いた三次元への応用

建築都市デザイン学科　2280210056-6　田中　雅也
（指導教員　山田悟史）

１．はじめに

長方形詰込み問題(Rectangle Packing Problem)は，与えられたn個の長方形を大きさが一定の二次元平面に重なりなく配置する問題である．古典的な組合せ最適化問題の一つで，鉄鋼・ガラス・繊維などの素材産業，集積回路の設計，倉庫やトラックの効率的な運用などに用いられる．大きな鉄板や布，段ボールから，必要に応じた大きさの製品に切り分ける問題や，一定の容器により多くの荷物を詰め込む問題といった多くの工学的応用分野の問題で活用されている．しかし，この問題は NP 困難であることが知られており，大規模な問題例に対して厳密な解を求めることは困難である．これまで，厳密な解を求める解法から，実用的な解を求める解法など，多種多様な解法が提案されてきた．本研究では，三次元への応用があるため，単純なルールに基づいたアルゴリズムであり，そのため長方形以外の詰込み問題にも応用できる Bottom-Left 法(以下, BL 法)に注目した．

BL 法とは，長方形詰込み問題に対する基本的な構築型解法の一つである．BL 法は，空の二次元矩形領域から始めて，長方形に対して与えられた順序どおりに一つずつ図形を配置していくという貪欲法である．図形を配置する場所は，その図形を配置しようとする時点において配置可能な場所の中の最も下の位置で，そのような場所が複数ある場合は，その中で最も左の場所に配置する．この解法の計算時間は$O(n^3)$程度で，解の精度は少なくとも最適解の 3 倍以内である*1)が，たいていの場合，$O(n^2)$の計算時間で最適解の 20%以内の解を出力する．

本研究では, BL 法を用いて，長方形詰込み問題を Python で近似的に解を求めることができることを示し，この問題を三次元に拡張し，直方体詰込み問題を Rhinoceros+Grasshopper で解くことを目的とした．

２．　定式化

簡単に長方形詰込み問題を定式化する．本研究はストリップパッキング問題を採用した．

幅（W: *固定*）と高さ（H: *可変*）という長方形領域と，n個の長方形群Iについて考える．この時，Iの長方形に対してid，幅（w），高さ（h），座標（x,y）を表す変数を定義する．

本問題の目的は長方形領域内にすべての長方形を重なりなく詰め込むという制約条件の下で，領域の高さHを最小化することである．次がその定式化である．

最小化　H

制約条件　$0 \le x_i \le W - w_i \ (i = 1,2,\cdots,n)$

$0 \le y_i \le H - h_i \ (i = 1,2,\cdots,n)$

長方形は互いに重ならない

w_i, h_iは非負である

この問題の解法として BL 法などがある．

３．実装方法

まず，長方形詰込み問題に対する BL 法の Python での実装を行う．この実装では，あらかじめすべての長方形の高さと幅を設定し，ナンバリングする．そして，長方形領域（幅のみを設定）に重なりを判定しながら長方形を順番に詰め込んでいく．この時，番号が若い順になるべく下，同じ高さであればできる限り左に詰め込むということを繰り返す．

本研究で BL 法は，順序を入れ替えることで解が大きく変わるということを利用して簡単な最適化を行った．ランダムに入れ替えたときに領域の高さが最も低いものを疑似最適解とする方法や，隣のものと順序を入れ替えることを低くならなくなるまで繰り返す，疑似的な勾配降下法を試した．

次に，これを三次元に拡張し，直方体詰込み問題に適用して，Rhinoceros+Grasshopper で実装した．この問題では，与えられたm個の直方体を幅W，奥行きD，高さH（可変）の直方体の容器に重なりなく配置するとき，その容器の高さHをできるだけ小さくする．この問題に対する BL 法は，直方体に対して与えられた順序の先頭から順番に一つずつなるべく底に，同じ高さであればできる限り手前に，さらに同じ奥行であれば左に詰めて配置していく．

３．結果

Grasshopper と Python で異なるコードの書き方をしたが，内容的にはほぼ同じ内容である．実装環境は，i5-1135G7 であり，Python3 で Anaconda3 の Jupyter Notebook と，Rhinoceros7 上での実行である．

Implementation of the rectangle packing problem with the bottom-left algorithm in Python and its application to three-dimensional shapes using Grasshopper

Masaya Tanaka

３．1. Python での実装の結果

使用する主な Python ライブラリは matplotlib, pandas, nympy, itertools である. matplotlib は図形の描画に用い, pandas は dataframe の処理に使用した. itertools は配置する座標を設定する際の組み合わせ順列を求める際に用いた. また, 実際に配置する長方形の数は 20 である.

```
import matplotlib.pyplot as plt
from matplotlib import patches
import math
import numpy as np
import random
import pandas as pd
import itertools
from IPython.display import clear_output
import time
%matplotlib inline

seed = 0
random.seed(seed)
np.random.seed(seed)

W = 4
B = 20
V = 20
wh = np.random.uniform(low=0.5, high=1.5, size=(B, 2))
wh = wh*(math.sqrt(V/np.sum(wh[:,0]*wh[:,1])))
xy = [[0,0]]*B

wh_df = pd.DataFrame(data=(wh),columns=("w","h"))
xy_df = pd.DataFrame(data=(xy),columns=("x","y"))
S_df = wh_df["w"]*wh_df["h"]
Rect = pd.concat([wh_df,xy_df],axis=1)
Rect["x+w"] = Rect["x"]+Rect["w"]
Rect["y+h"] = Rect["y"]+Rect["h"]

def Rect_place(rect,n):
    global ax,fig
    ax.grid()
    colors="#"+''.join([random.choice¥
     ('0123456789ABCDEF') for i in range(3)])
    r = patches.Rectangle( xy=(rect.loc[n]["x"],¥
    rect.loc[n]["y"]) , width=rect.loc[n]["w"], ¥
    height=rect.loc[n]["h"], color=colors)
    ax.add_patch(r)

def overlap(rect,n):
    result = []
    for i in range(n):
        xn = rect.loc[n]["x"]
        xi = rect.loc[i]["x"]
        wn = rect.loc[n]["w"]
        wi = rect.loc[i]["w"]
        yn = rect.loc[n]["y"]
        yi = rect.loc[i]["y"]
        hn = rect.loc[n]["h"]
        hi = rect.loc[i]["h"]
        result.append(¥
        not((max(xn,xi)<min(xn+wn,xi+wi))¥
        and(max(yn,yi)<min(yn+hn,yi+hi))))
    return all(result)

def Out(rect,n):
    xn = rect.loc[n]["x"]
    yn = rect.loc[n]["y"]
    wn = rect.loc[n]["w"]
    hn = rect.loc[n]["h"]
    return xn>=0 and yn>=0 and xn+wn<=W
```

```
def place_coordinate(rect,n):
    x_coordinate = pd.concat([rect.loc[:,"x+w"],¥
    rect.loc[:,"x"]],axis=0).drop_duplicates()
    y_coordinate = pd.concat([rect.loc[:,"y"], ¥
    rect.loc[:,"y+h"]],axis=0).drop_duplicates()
    coordinate = pd.DataFrame¥
    (itertools.product¥
    (x_coordinate,y_coordinate)).rename¥
    (columns={0:'x',1:'y'})
    coordinate_sort = coordinate.sort_values¥
    (by=["y","x"], ascending=True)
    coordinate_sort = coordinate_sort.set_axis¥
    (list(np.arange(0, len(coordinate))), axis=0)
    i = 0
    while True:
        rect.at[n,"x"] = coordinate_sort.loc[i]["x"]
        rect.at[n,"y"] = coordinate_sort.loc[i]["y"]
        if overlap(rect,n) and Out(rect,n):
            break
        if i == len(coordinate):
            break
        i += 1
    rect["x+w"] = rect["x"]+rect["w"]
    rect["y+h"] = rect["y"]+rect["h"]
```

注:¥は改行記号である

以上が定義部分である.

前半部分はライブラリのインポートや数値の設定, dataframe の作成, 長方形を領域内に配置する関数を生成している. そして, 後半部分は BL 法の核となる 3 つの関数を定義している. 以下がその関数の説明である.

Overlap 関数：長方形同士が重なっていないかを確認する. もし重なっていた場合 True を返す.

Out 関数：領域内に収まっているかを確認する. Ryou 域外に飛び出ていれば, True を返す.

Place_coodinate 関数：長方形を配置する座標を求める. 配置の候補となる座標を各長方形の座標の組み合わせをもとに全て列挙し, y, xの順にソートした後実際に長方形を配置する. その後, overlap 関数と out 関数を使用してその座標が実際に配置できるかを判定し, 配置できるまで繰り返す.

以下は, 実行と描画のコードである.

```
Rect2 = Rect.copy()
for j in range(B):
    place_coordinate(Rect2,j)

Height = round(Rect2.max()["y+h"],2)
H = math.ceil(Height)

fig, ax = plt.subplots(figsize=(W,H))
ax.set_xticks([0, W])
ax.set_yticks([0, H])
ax.axhline(Height)
plt.text(4.1,Height,Height,size='10')
ax.grid()

for n in range(B):
    Rect_place(Rect2,n)

print(f"高さ：{Height}")
plt.show()
```

実行結果は Fig. 1 のようになる．同環境では，実行時間は約 7 秒となっている．色分けはランダムに行っている．

次に，順序をランダムに入れ替えてその中から最も低い解を求めた．

```
H_lst = []
j = 100
for i in range(j):
    clear_output(True)
    print(i+1,"/",j)
    Rect4 = Rect.sample¥
    (frac=1, ignore_index=True,random_state=i)
    for k in range(B):
        place_coordinate(Rect4,k)
    H_lst.append(Rect4.max()["y+h"])
clear_output()
num = H_lst.index(min(H_lst))

Rect4 = Rect.sample¥
(frac=1, ignore_index=True,random_state=num)

for j in range(B):
    place_coordinate(Rect4,j)
```

結果は Fig. 2 のようになる．

順序をランダムに 100 個挙げるだけで高さH=6.54 からH = 5.62まで下げることができた．計算時間は約 1872 秒となっている．最後に，疑似的な勾配降下法である，隣同士を入れ替えて，低くならなくなるまで繰り返すという方法で行った．開始地点は，ランダムに行った時に得られた近似解からである．

```
Rect5 = Rect4.copy()
count = 0
while True:
    count += 1
    for k in range(B):
        place_coordinate(Rect5,k)
    H_lst = []
    H1 = Rect5.max()["y+h"]
    H_lst.append(H1)
    for i in range(B-1):
        Rect_test = Rect5.copy()
        Rect_test = replace(Rect_test,i,i+1)
        for k in range(B):
            place_coordinate(Rect_test,k)
        H2 = Rect_test.max()["y+h"]
        H_lst.append(H2)
        print(i+1,H_lst)
        clear_output(True)
    num = H_lst.index(min(H_lst))

    if num == 0:
        break
    else:
        Rect5 = replace(Rect5,num-1,num)
```

結果は, Fig. 3 のようになる．H=5.62 からH = 5.56と微減した．計算時間は約 471 秒である．1 回隣のものと入れ替えるだけの結果となった．

局所最適解は厳密な最適解ではないが，ランダムなものの近似解から開始したので，今できうる限りの近似最適解といえる．

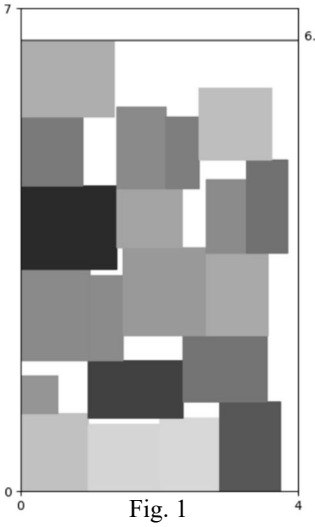

Fig. 1

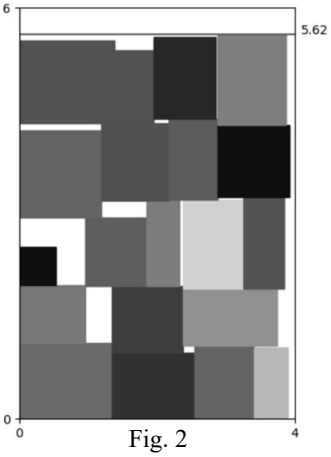

Fig. 2

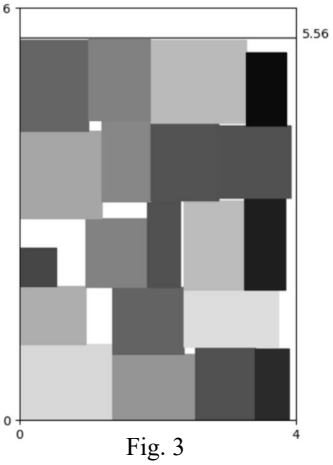

Fig. 3

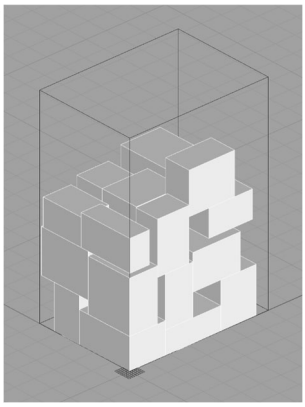
Fig. 4

３．２．Grasshopper での実装の結果

実装にあたって, Grasshopper のプラグインとして回帰処理を行える Anemone を利用した. Grasshopper 内でも，実装に困難だと思われる部分については, GhPython を用いて処理した．結果は Fig. 4 のようになる. Grasshopper でも 20 個の直方体を設置した．計算時間は約 250 秒である．

アルゴリズムはほぼ Python と同じであり，高さと幅に加えて奥行が追加されただけである．しかし，処理の方法は Python の場合と異なる点もいくつかあり, Python では for 文で一つ一つ配置して，重なりの判定を行っていたが，Grasshopper は一度に配置することができるためこの点において，計算は早い．しかし，前者は数値の状態で計算を行っていたのに対し，実際に直方体を配置して衝突判定を行ったり，あるいは点群に分解したりとひと手間かかってしまっていることや, 3 次元になると頂点の数が 2 次元の時より 2 倍に増え，配置する頂点の数も指数関数的に増えているため，計算時間が約 35 倍に増えたと考えられる．また, Galapagos などの最適化コンポーネントと Anemone の相性が悪いため，最適化は行っていない．一度にかかる計算時間が多い場合，うまく動かない．

４．まとめ

長方形詰込み問題に対する BL 法の Python での実装と，その最適化手法の提案を行った. BL 法は，長方形の順序に依存するため，順序の決め方や改善法に工夫が必要であることを示した. また，ランダムに順序を入れ替えたり，隣のものと入れ替えることを繰り返すことで，解の品質を向上させることができた.

BL 法を三次元に拡張して， 直方体詰込み問題に適用し，Rhinoceros+Grasshopper での実装を行った． 直方体詰込み問題では，幅，奥行き，高さの 3 つの次元を考慮する必要があるため， 計算時間や配置の候補が増加するという課題があった． しかし， BL 法はそのシンプルさと効率性から，この問題にも適用できることを示した.

BL 法は， まだまだ発展的な問題を含んでいる． 例えば，長方形や直方体以外の図形，レクトリニア図形や円にも応用すること*2)や， 本研究では， 図形の配置において， 回転を許さないという制約を課しているが， それらを緩和し， 図形の回転角度も変数化することで， より効率的な配置を得ることなどがあげられる． これから， これらの問題にも積極的に取り組んでいきたい.

参考文献

1）今掘 慎治，梅谷 俊治：切出し・詰込み問題とその応用（2）長方形詰込み問題，日本オペレーションズ・リサーチ学会, 5 月号, pp.335-340, 2005 年

2）今堀 慎治，胡 艶楠，橋本 英樹，柳浦 睦憲：Python による図形詰込みアルゴリズム入門，オペレーションズ・リサーチ, 12 月号, pp.762-769, 2018 年

3）並木 誠，久保幹雄：Python による数理最適化入門，朝倉書店, 2018 年

4）小林 和博：錐最適化・整数最適化・ネットワークモデルの組み合わせによる最適化問題入門，近代科学社, 2020 年

5）寒野 善博，駒木 文保：最適化手法入門(データサイエンス入門シリーズ)，講談社, 2019 年

6）須藤秋良：スッキリわかる Python による機械学習入門(スッキリわかる入門シリーズ)，株式会社フレアリンク, 2020 年