Pythonによる図形詰込みアルゴリズム入門

今堀 慎治，胡 艶楠，橋本 英樹，柳浦 睦憲

最適化問題の中でも図形詰込み問題は，解（図形の配置）を描画することで視覚的にその状況を把握することができるため初学者にも取り組みやすい問題であろう．本稿では，Python による図形詰込みを紹介する．まず 2 次元の長方形詰込み問題に対して提案された基本的なアルゴリズムである bottom-left 法（BL 法）を紹介する．BL 法は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図形詰込み問題にも応用することが可能である．本稿ではそのような例として，円の容器に円の図形を配置する問題や，レクトリニア図形や直方体の詰込み問題への BL 法の適用を示す．

キーワード：bottom-left 法，図形配置，Python 言語，長方形，レクトリニア図形，円，直方体

1. はじめに

最適化問題の中でも図形詰込み問題は，解（図形の配置）を描画することで視覚的にその状況を把握することができるため初学者にも取り組みやすい問題であろう．本稿では，Python による図形詰込み問題に対する基本的なアルゴリズムを紹介する．Python は，C 言語などと比べると非常に簡潔にプログラミングでき，また，図形を簡単に描画できるライブラリなど多様なツールが存在するため，非常に短期間でアルゴリズムを実装することができる．

詰込み問題とは，配置すべきもの（製品と呼ぶ）の集合と配置される空間（容器と呼ぶ）が与えられたとき，製品を容器内に，さまざまな制約の下で効率よく配置する問題である．この問題は，古典的な組合せ最適化問題の一つであり，これまでに多くの研究がなされてきた．計算の複雑度という観点から見ると，ほぼすべての詰込み問題は NP 困難に分類され，多項式時間で厳密な解を求める手法はおそらく存在しない．詰込み問題の中でも工学的に重要な意味をもつ，長方形詰込み問題，多角形詰込み問題，直方体詰込み問題に対する研究が盛んである．これらの問題の直接的な応用先として，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が挙げられる．

2 次元の図形詰込み問題においては，長方形はその扱いが容易であるため，これまで長方形詰込み問題に対して多くの研究が行われてきた．その結果，大規模な問題例にも適用できる高速なアルゴリズムも提案されている．代表的な構築型解法に bottom-left 法（BL 法）[1] と best-fit 法（BF 法）[2] がある．BL 法も BF 法も容器に何も配置されていない状態から始めて，長方形を一つずつ配置していき，最後に解を得るという構築タイプの解法である．BL 法では，長方形の順序があらかじめ与えられ，その順序で長方形を一つずつ配置する．best-fit 法はすでに配置されている状態を見て，次に配置する長方形を動的に選ぶため，BL 法より少し複雑な解法である．

本稿では，BL 法に焦点を絞って，いくつかの詰込み問題に対する Python の実装を紹介する．まず長方形詰込み問題に対する BL 法を紹介する．さらに一般の図形詰込み問題に対する応用として，レクトリニア図形や円などの詰込み問題への BL 法の適用を示す．

2. 長方形詰込み問題

長方形詰込み問題とは，与えられた n 個の長方形を幅 W の長方形の容器に重複なく配置するとき，その容器の高さ H をできるだけ小さくする問題である（図 1）．各長方形 i の幅を w_i，高さを h_i とする．長方形の回転を許して配置する場合もあるが，本稿では長方形の回転は許さないものとする．

この問題を整数計画問題として定式化しよう．長方形 i の左下の点の座標を (x_i, y_i) とする．長方形 i が

いまほり しんじ
中央大学理工学部
〒 112–8551 東京都文京区春日 1–13–27
imahori@ise.chuo-u.ac.jp
こ えんなん，やぎうら むつのり
名古屋大学大学院情報学研究科
〒 464–8601 愛知県名古屋市千種区不老町
yannanhu@nagoya-u.jp
yagiura@nagoya-u.jp
はしもと ひでき
東京海洋大学海洋工学部
〒 135–8533 東京都江東区越中島 2–1–6
hhashi0@kaiyodai.ac.jp

 Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

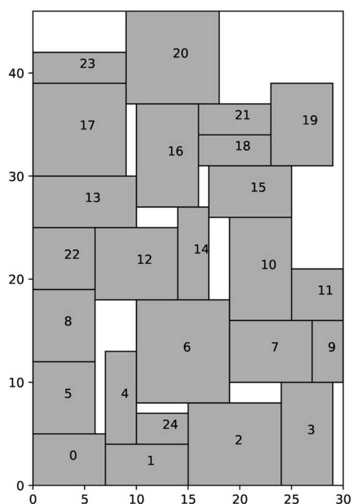

図 1　長方形詰込み問題の配置例

j の左にあるとき $u_{ij}=1$，そうでないときは $u_{ij}=0$ とする．また，長方形 i が j の下にあるとき $v_{ij}=1$，そうでないときは $v_{ij}=0$ とする．長方形詰込み問題は次のように定式化できる:

$$
\begin{aligned}
&\min H\\
&\text{s.t. } x_i+w_i \le x_j + W(1-u_{ij}),\ i,j\in\{1,\dots,n\}\\
&\quad y_i+h_i \le y_j + H_{\text{UB}}(1-v_{ij}),\ i,j\in\{1,\dots,n\}\\
&\quad 0\le x_i \le W-w_i,\quad i\in\{1,\dots,n\}\\
&\quad 0\le y_i \le H-h_i,\quad i\in\{1,\dots,n\}\\
&\quad u_{ij}+u_{ji}+v_{ij}+v_{ji}\ge 1,\ i,j\in\{1,\dots,n\}\\
&\quad u_{ij},v_{ij}\in\{0,1\},\ i,j\in\{1,\dots,n\}.
\end{aligned}
$$

ここで，H_{UB} は H の上界値（たとえば $\sum_{i=1}^{n} h_i$）に設定すればよい．制約の第 1 式（第 2 式）は $u_{ij}=1$ $(v_{ij}=1)$ ならば長方形 i を j の左（下）に配置することを，第 3 式（第 4 式）は各長方形 i が容器の左にも右にも（下にも上にも）はみ出さないことを意味する．また，第 5 式はどの長方形対に対しても上下左右の位置関係のうち少なくとも一つを指定しなければならないことを表す．

この問題は NP 困難であることが知られており，次節で述べるように整数計画問題に対する汎用ソルバーで求解できる問題例の規模は小さい．

2.1　PuLP による求解

試しに Python のライブラリ PuLP[1]で長方形詰込み問題を解いてみよう．簡単にいろいろ試せるのが Python のよいところである．上記の定式化は PuLP で以下のように実現できる．文法を知らなくとも，なんとなく書いてあることが推察できてしまうのも Python のよいところである．たとえば「`for i in range(n)`」と書けば `i` には `0` から `n-1` までの整数が順に代入される．また，“\” は次行に続くことを表す．PuLP については本特集の別の記事 [3] で解説されているので，ここでは省略する．このプログラムでは，前半部分で `LpProblem` でモデルオブジェクト `m` を生成し，`LpVariable` で変数オブジェクトを生成している．そして後半部分で目的関数と制約を「`+=`」演算子によりモデル `m` に追加している．

```
from pulp import *
m = LpProblem(sense=LpMinimize)

x=[LpVariable("x%d" %i,lowBound=0)\
   for i in range(n)]
y=[LpVariable("y%d" %i,lowBound=0)\
   for i in range(n)]
u=[[LpVariable("u%d%d" %(i,j),cat=LpBinary)\
   for j in range(n)]\
   for i in range(n)]
v=[[LpVariable("v%d%d" %(i,j),cat=LpBinary)\
   for j in range(n)]\
   for i in range(n)]
H=LpVariable("H")

m += H
for i in range(n):
    for j in range(n):
        m += x[i]+w[i] <= \
             x[j]+W*(1-u[i][j])
        m += y[i]+h[i] <= \
             y[j]+UB*(1-v[i][j])
        if i < j:
            m += u[i][j]+u[j][i]\
                +v[i][j]+v[j][i] >= 1
for i in range(n):
    m += x[i] <= W-w[i]
    m += y[i] <= H-h[i]

m.solve()
```

筆者の PC (Intel Core i5 2.9 GHz) で試してみたところ，8 個の長方形の問題例に対して 61 秒で最適解が得られたが，これより長方形の数が少しでも多くなると求解にかかる時間が急激に増大する．（もちろんこれが厳密解法の限界ではない．この問題に特化した厳密解法の研究もあり，より大きな問題例に対して厳密解が得られている [4].）

2.2　Python による図形の描画

次に Python による図形の描画方法を紹介する．領域 $[0,W]\times[0,H]$ に長方形と円を描画しよう．ここでは描画に Python のライブラリ matplotlib を使用する．以下のプログラムのはじめの 3 行は描画に必要な初期設定である（詳細は略す）．

描画領域の設定は，`ax.set_xlim([0,W])` と `ax.set_ylim([0,H])` のようにできる．幅 w，高さ

[1] https://pythonhosted.org/PuLP/

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

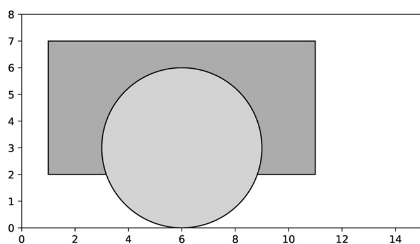

図 2 Python による図形描画

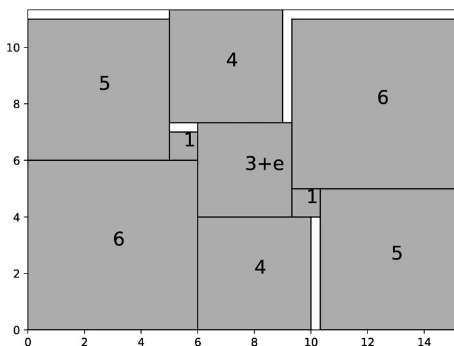

図 3 BL 法では最適解が得られない問題例

h の長方形を左下の角が座標 (x, y) になるように配置するには Rectangle を使って次のように書けばよい（fc と ec はそれぞれ長方形内部と辺の色を表す）．円を描く場合も同じようにして Circle を使って円の中心と半径を指定すれば描画できる．描画した画像は savefig を使ってさまざまな画像フォーマットで保存することができる．次のプログラムの実行結果を図 2 に示す．

```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111,aspect='equal')
W,H = 15,8
ax.set_xlim([0,W])
ax.set_ylim([0,H])

x,y,w,h = 1,2,10,5
rect = plt.Rectangle((x,y),w,h,\
                     fc="burlywood",\
                     ec="black")
ax.add_patch(rect)

center,r = (6,3),3
circ = plt.Circle(center,r,\
                  fc="lightgray",\
                  ec="black")
ax.add_patch(circ)

fig.savefig("picture.eps")
```

3. bottom-left 法

bottom-left 法 [1] は長方形詰込み問題に対して提案された基本的なアルゴリズムの一つである．アルゴリズムは単純で，空の容器から始めて，図形に対して与えられた順序の先頭から順番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．

図 1 は BL 法によって得られた配置例である．この例では長方形の番号は与えられた順序の番号であり，長方形 0 が最初に左下の角に配置されている．

BL 法は長方形に対して与えられた順序により得られる配置が異なる．長方形の幅や面積の降順に並べた順序などがしばしば用いられる．また，局所探索法などの探索型手法を用いて順序を決定することでよい配置を求めることも可能である．ただし，すべての順序を生成し，その各々に対して BL 法を適用しても最適解が得られないことがある．図 3 の問題例は，長方形がすべて正方形で内部の数字は 1 辺の長さを表している（ただし $0 < e < 1$）．容器の幅は $W = 15 + e$ である．この問題例に対して，最適解の高さは $11 + e$ であるが，BL 法によってこの配置を得ることはできない．

3.1 節では，まず，Python による BL 法の単純な実装を示し，次に 3.2 節でデータ構造を工夫した実装を紹介する．

3.1 単純な実装

ここでは BL 法の単純な実装を示す．いくつかの長方形が配置された容器に新しい長方形を配置することを考える．新しい長方形 r を容器の中の場所 p に配置したとき，既配置の長方形と重複なく，かつ r を場所 p から下または左に動かそうとしても既配置の長方形と重複または容器からはみ出るため動かすことができない場所のことを BL 実行可能点 (BL feasible point) と呼ぶ．このような BL 実行可能点の中で最も下の位置で最も左にある場所を BL 点と呼ぶ．BL 法とは，長方形を与えられた順番にその BL 点に配置する方法にほかならない．

ここで紹介する BL 法の実装では，BL 実行可能点となる可能性のある点を生成し，その各々に長方形を重複なく配置可能であるかを判定することで，BL 実行可能点を求め，それらから BL 点を求める．BL 実行可能点となる可能性のある点とは，長方形 r をその場所から下にも左にも動かせない点のことである．そのような点の各々は，下に動かそうとしたときにぶつかるもの（長方形または容器）と左に動かそうとしたときに

 Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

ぶつかるもののペアにより一意に定まる．そのようなペアは $O(n^2)$ 個あり，そのすべてを調べればよい．この実装の計算時間は，ペアの個数が $O(n^2)$，一つのペアに対して重複なく配置できるか判定するのに $O(n)$ 時間，つまり，既配置の長方形に対して，新しい長方形の BL 点を探索するのに $O(n^3)$ 時間かかる．BL 法ではこの操作を n 回繰り返すので，合計で $O(n^4)$ 時間となる．

BL 法の単純な実装は，次のように三つの関数で実現できる．プログラム中の w と h は各長方形の幅と高さが格納されているリスト（C 言語の配列のようなもの）である．

- BL_method(w,h,W): 長方形集合の幅 w と高さ h，容器の幅 W が与えられたとき，BL 法を実行し，各長方形の座標をリスト x と y に保存する関数．最初の for 文では長方形 i を一つずつ配置していく．bl_candidates() が生成する i を置く候補点をまず cand に格納し，その中から容器からはみ出さず，既配置の長方形と重なりなく置ける場所のみをリスト blfp に格納する．blfp の中の最も下に位置する点の中で最も左の点は，（Y 座標，X 座標）の辞書式順序で最も小さい点であるので，key をそのように設定して min で計算する．最後に得られた i の配置位置をリスト x と y の最後（i 番目）に格納する．
- bl_candidates(i,x,y,w,h): 長方形 0, 1, . . . , i-1 が既配置のとき，長方形 i の BL 実行可能点の候補となる点集合のリスト cand を返す関数．cand の計算は，まず容器の左下の点 (0, 0) を一つ目の要素とする．次に，既配置の長方形 j と k に対して，j の右辺と k の上辺の交点，および k の右辺と j の上辺の交点をリスト cand に追加する．さらに，容器の左辺と既配置の長方形 j の上辺の交点，および j の右辺と容器の底辺の交点を追加する．
- is_feas(i,p,x,y,w,h,W): 長方形 0, 1, . . . , i-1 が既配置のとき，長方形 i を座標 p に重複なく配置可能かどうかを判定する関数．最初の if 文では，p に i を置くと容器の左または右にはみ出す場合は「置けない」ことを意味する False を返して終了する．次の for 文で，p に i を置くと既配置の長方形 j と重複する場合は同様に False を返して終了する．最後に，上記のようなものがなければ「置ける」ことを意味する True を返して終了する．

```
def bl_candidates(i,x,y,w,h):
    cand=[(0,0)]
    for j in range(i):
        for k in range(j):
            cand += [(x[j]+w[j],y[k]+h[k])]
            cand += [(x[k]+w[k],y[j]+h[j])]
    for j in range(i):
        cand += [(0,y[j]+h[j])]
        cand += [(x[j]+w[j],0)]
    return cand

def is_feas(i,p,x,y,w,h,W):
    if p[0] < 0 or W < p[0]+w[i]:
        return False
    for j in range(i):
        if max(p[0],x[j]) < \
        min(p[0]+w[i],x[j]+w[j]):
            if max(p[1],y[j]) < \
            min(p[1]+h[i],y[j]+h[j]):
                return False
    return True

def BL_method(w,h,W):
    x,y=[],[]
    for i in range(len(w)):
        blfp=[]
        cand = bl_candidates(i,x,y,w,h)
        for p in cand:
            if is_feas(i,p,x,y,w,h,W):
                blfp += [p]
        min_p = min(blfp,key=\
                lambda v:(v[1],v[0]))
        x += [min_p[0]]
        y += [min_p[1]]
    return x,y

w=[9, 4, 4, 7, 5]
h=[4, 10, 9, 9, 10]
W=20

x,y=BL_method(w,h,W)
```

3.2 高度な実装

BL 法において最も時間がかかるのは BL 点の探索であり，それに対してデータ構造を工夫した高速な実装が知られている．

3.2.1 no-fit polygon

no-fit polygon (NFP) [5] とは，平面上で多角形の重なりを判定する方法である．多角形 P と Q が与えられ，P の配置が固定されているとする．このとき，P と Q が重なりをもつような Q の配置位置の集合を P に対する Q の NFP と呼び，$\mathrm{NFP}(P, Q)$ と表す．P と Q がともに長方形の場合，$\mathrm{NFP}(P, Q)$ は Q を P と接するように平行移動させたときの Q の配置位置（本稿では左下の頂点の座標）の軌跡の内部領域である．図 4 は NFP の例であり，太線が NFP の境界を表す．

3.2.2 2 次元上の BL 点発見法

Find2D-BL 法 [6] は，NFP を用いて 2 次元上の BL 点を発見するアルゴリズムである．以下では Find2D-BL 法の考え方の概要と，BL 点の計算に要する時間を紹介する．

 Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

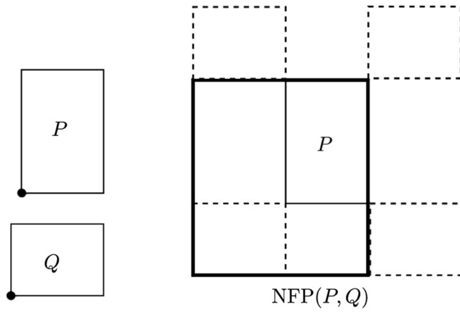

図 4　NFP の例

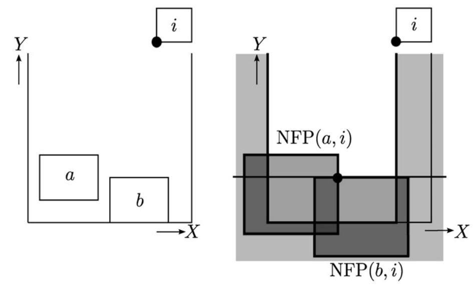

図 5　2 次元の BL 点の例

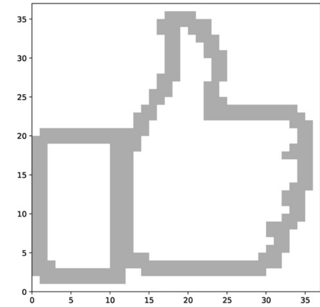
図 6　レクトリニア図形による近似

既配置の長方形すべてに対し長方形 i の NFP を作成すると，i の座標がそれら NFP のいずれの内部にも含まれないならば，既配置の長方形のいずれとも重なることなく i をその位置に置くことができる．しかし，そのような位置でも，i が容器からはみ出してしまって，配置できない場合がある．よって，容器に対しても i の NFP を考える．i を容器の内側に接するように平行移動させたときの i の座標の軌跡の外部領域が，容器に対する i の NFP となる．これらを用いると，既配置の長方形に対する i の NFP と容器に対する i の NFP のいずれにも i の座標が重ならない位置であれば，i を既配置の長方形に重なることなく，しかも容器からはみ出すことなく配置できることがわかる．

Find2D-BL 法は，走査線を用いてこのような点を発見するという考え方に基づいている．X 軸に平行な走査線を考え，これを容器の底から Y 軸の上方向に向かって動かす．このとき，走査線上の任意の点における NFP の重複の数がわかるようなデータ構造を維持しておく．走査線上で NFP の重複数が 0 になる位置が初めて現れたとき，走査線上のそのような位置の中で X 座標の値が最も小さい位置が BL 点となる．図 5 の左の図はいくつかの既配置の長方形（a と b）とこれから置こうとする長方形 i を表し，右の図は既配置の長方形に対する i の NFP と容器に対する i の NFP（太線が NFP の境界），および BL 点（中央の黒丸）を示している．

NFP を利用して重なりの有無を判定すると，BL 法の計算時間は単純な実装でも $\mathrm{O}(n^4)$ から $\mathrm{O}(n^3)$ に改善できる．さらに，Find2D-BL 法は $\mathrm{O}(n \log n)$ 時間で新しい長方形の BL 点を見つけることができるため，Find2D-BL 法を用いれば，BL 法の計算時間は $\mathrm{O}(n^3)$ からさらに $\mathrm{O}(n^2 \log n)$ に改善できる．

4.　2 次元図形詰込みへの応用

BL 法で用いられた，図形を下にも左にも動かせない場所に配置するというルールは，長方形以外の図形の配置に対しても応用することができる．

ここでは，レクトリニア図形を長方形の容器に配置する問題と，円を円の容器に配置する問題に対する応用を紹介する．

4.1　レクトリニア図形の詰込み

レクトリニア図形は縦の線分と横の線分で囲まれた図形である．長方形を少し一般化した概念で，アルゴリズム設計において長方形がもっていた扱いやすい性質をもちつつ，応用の幅は格段に広がる．たとえば曲線を含むような一般の図形もレクトリニア図形で近似することが可能なので（図 6），近似的に一般の図形の詰込みに応用することもできる．

レクトリニア図形は長方形の集合として表現できる．レクトリニア図形の頂点の数を r とすると，高々 $r-1$ 個の長方形の集合として表現できることは容易にわかる．ここでは，レクトリニア図形は長方形の集合として表されるものとし，レクトリニア図形 i を表現する長方形の数を r_i とおく．

4.1.1　単純な実装

レクトリニア図形の BL 点を考えるにあたって，ま

 Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

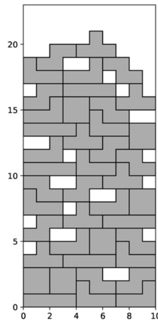

図 7　レクトリニア図形の詰込み

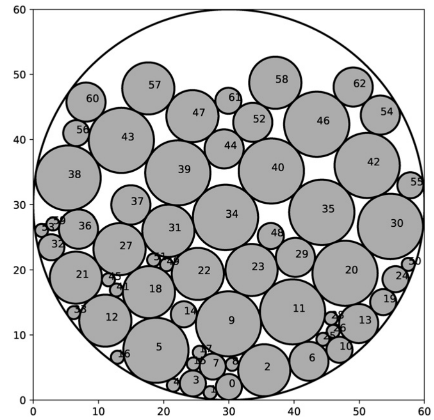

図 8　円の容器に円を配置

ずはレクトリニア図形を下に動かせない状況を考えてみよう．それは，すでに配置したレクトリニア図形を構成しているいずれかの長方形もしくは容器と，これから配置しようとしているレクトリニア図形を構成するいずれかの長方形が接するという状況である．左に動かせない状況も同様である．つまり，すでに配置しているレクトリニア図形を構成する長方形（あるいは容器）のうちの二つと，これから配置しようとしているレクトリニア図形を構成する長方形のうちの二つを決めれば，配置する座標が一意に定まる．

以上から次のような単純な実装ができることがわかる．$R = \sum_{i=1}^{n} r_i$ とおく．レクトリニア図形 i を配置しようとするとき，BL 実行可能点の候補は，上述のように基本的に四つの長方形を決めればその座標が一意に定まるので，その数は $\mathrm{O}(R^2 r_i^2)$ となる．一つの BL 実行可能点の候補に対して，重複（実行可能性）の判定は各長方形が重複するかどうかを判定すればよいので，$\mathrm{O}(Rr_i)$ 時間で計算できる．したがって，合計で $\mathrm{O}(\sum_{i=1}^{n} R^2 r_i^2 R r_i) = \mathrm{O}(R^6)$ 時間となる．

図 7 は，BL 法のこのような単純な実装により得られた配置である．前述の筆者の PC で Python で実装したプログラムを実行したところ，この 42 個の図形の問題例に対して配置を得るのに 47 秒かかった．

4.1.2　高度な実装

3.2 節で紹介した長方形詰込み問題に対する NFP と Find2D-BL 法の考え方を利用して，前節で説明した単純な実装から大幅に計算時間を改善できる [7]．以下ではその高度な実装の概略を示す．

BL 法は構築型解法であり，容器にいったん置いた図形を消したり移動したりしない．この性質を利用し，相異なる形状ごとに，既配置の図形に対して，その形状の NFP の集合を保存する．そして，図形を配置するたびに NFP の集合を更新する．なお，ここでは合同でしかも配置する際の向きが同じ図形を同じ形状とみなしている．形状が同じであれば NFP の集合は同じであるため，同じ形状の図形が複数含まれている問題例では，形状の同じ複数の図形に対して，このような NFP の集合を一つ保持すればよい．また，どんな形状の BL 点も，新しい図形を置くことで下に下がることはない，つまり，走査線をその形状の前回の BL 点の高さから上に向かって走査すればよい．これらのアイデアを用いて，各形状に対し，現在構築中の解に対応する NFP の集合を保持し，解の更新に応じて逐次修正していくと，高速化できる．形状ごとに，NFP の集合を平衡探索木，走査線をヒープで管理し，図形を配置するたびに各平衡探索木とヒープを更新する．相異なる形状すべてに対する r_i の合計を r とすると，BL 法の計算時間は $\mathrm{O}(Rr \log R)$ となる．

図 7 の問題例に対して，C 言語により実装されたプログラムを筆者の PC で実行したところ，同じ配置を得るのに 0.0178 秒であった．図形の数が 1 万個程度の問題例に対しても，Intel Core i5, 2.3 GHz, 3 MB cache, 4 GB memory の PC で，2 秒以内に解を得ることが可能である．

4.2　円の詰込み

円の図形を円の容器に配置する問題を考える（図 8）．この問題についてもまず BL 実行可能点の候補について考えよう．容器に接する場所に配置する場合以外は，配置しようとしている円とすでに配置されている二つの円の計三つの円から配置する座標を一意に定めることができる．つまり，円 i の半径を d_i とすると，すでに配置されている二つの円 i と j からそれぞれ $d_i + d_k$ と $d_j + d_k$ の距離にある点を求めればよい．このような計算はいわゆる三辺測量と同様に解析的に計算することができる．解として 2 点の座標が得られるが，す

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

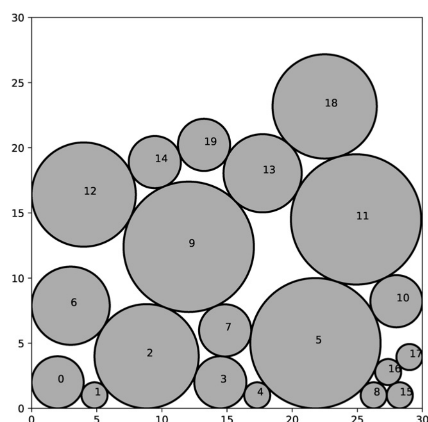

図 9　長方形の容器に円を配置

でに配置されている二つの円と下側と左側で接する点を使えばよい．

以上から次のような単純な実装ができる．新しい円を配置するときにその配置する座標の候補として $\mathrm{O}(n^2)$ 個の点があり，各点に対して重複判定に $\mathrm{O}(n)$ 時間かかる．この操作を n 回繰り返すので，全体としては長方形詰込み問題のときと同様に $\mathrm{O}(n^4)$ 時間となる．この単純な実装によって得られた配置例を図 8 に示す．

上記の実装は，容器の形状が長方形になっても微修正で対応することができる（図 9）．

5. 3 次元直方体詰込みへの応用

BL 法のさらなる応用として直方体詰込みを考える（図 10）．直方体詰込み問題にもいろいろあるが，ここでは与えられた n 個の直方体を幅 W と高さ H が固定された容器に重複なく配置するとき，その奥行き D を最小にすることを目的とする問題を考える．この問題に対しては，BL 法の自然な拡張として DBL 法がある．これは，BL 法と同様に容器が空の状態から始め，与えられた順序に従って直方体を一つずつ配置していく方法で，各直方体を配置する際には，その時点でその直方体を重なりなく置ける位置の中で

・最も奥
・そのような点が複数ある場合は最も下
・そのような点が複数ある場合は最も左

に配置する．このような配置位置を DBL 点と呼ぶ．

この方法を実現するうえで最も時間がかかるのは，各直方体の DBL 点を求める部分である．これを単純に行うと，直方体一つ当たり $\mathrm{O}(n^4)$ 時間かかる（直方体をそれ以上奥にも下にも左にも動かせない位置の中にそのような位置があることから，$\mathrm{O}(n^3)$ 個の候補のそれぞれに対して $\mathrm{O}(n)$ 時間かけてその位置にほかの直方体と重複なく配置できるかどうかを調べればよい）．これをより高速に実現するために，3.2 節で述べた BL 点を高速に求める方法が利用できる．配置しようとする直方体がそれ以上奥に動かせない位置の候補は，配置しようとする直方体を奥に動かそうとするとぶつかる面，すなわち容器の奥の面，あるいは既配置の直方体の手前の面を含む平面上にある．その各々に対して，配置しようとする直方体をその平面上に置くときに重複しうる直方体すべてをその平面上に射影して得られる長方形の配置に対して，配置しようとする直方体を射影した長方形の BL 点を求め（存在しない場合もある），その中の一番奥のものを出力すればよい．考慮すべき面の数は $\mathrm{O}(n)$ であり，その各々に対して Find2D-BL 法を用いて $\mathrm{O}(n \log n)$ 時間で BL 点を求めれば，一つの直方体の DBL 点を求めるのに要する時間は $\mathrm{O}(n^2 \log n)$ 時間である．

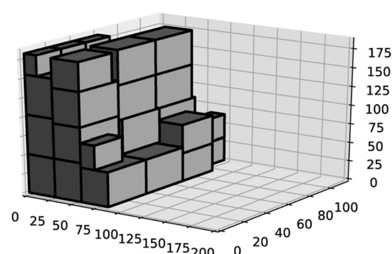

図 10　直方体配置

このような考え方を基本として，無駄な探索を省くさまざまなアイデアを組み合わせることで，DBL 法や 3 次元 best-fit 法の計算時間をさらに高速化する試みもあり，1 万個の直方体に対する配置が数分 (Xeon 3 GHz) で得られることが報告されている [8, 9]．

6. まとめ

本稿では Python を用いて図形詰込み問題における基本的なアルゴリズムである bottom-left 法（BL 法）を紹介した．BL 法は 2 次元の長方形詰込み問題に対して提案された手法であるが，そのアイデアは長方形以外の図形や 3 次元の物体の配置にも応用できる．本稿では BL 法をそのような詰込み問題に応用する方法についても述べた．さまざまな詰込み問題に対して構築型解法は有効であり，そのような解法を手軽に実装するうえで Python は強力なツールである．本稿がそのような問題に対する実践的解法を開発する際の一助となれば幸いである．

 Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.

謝辞　本研究は一部グローバル ATC (No.1005304) の助成を受けたものである．

参考文献

[1] B. S. Baker, E. G. Coffman Jr. and R. L. Rivest, "Orthogonal packings in two dimensions," *SIAM Journal on Computing*, **9**, pp. 846–855, 1980.

[2] E. K. Burke, R. Hellier, G. Kendall and G. Whitwell, "A new placement heuristic for the orthogonal stock-cutting problem," *Operations Research*, **54**, pp. 587–601, 2006.

[3] 斎藤努，"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，" オペレーションズ・リサーチ：経営の科学，**63**(12), pp. 784–790, 2018.

[4] J.-F. Cote, M. Dell'Amico and M. Iori, "Combinatorial Benders' cuts for the strip packing problem," *Operations Research*, **62**, pp. 643–661, 2014.

[5] R. C. Art, "An approach to the two dimensional irregular cutting stock problem," Ph.D. Thesis, Massachusetts Institute of Technology, 1966.

[6] S. Imahori, Y. Chien, Y. Tanaka and M. Yagiura, "Enumerating bottom-left stable positions for rectangle placements with overlap," *Journal of Operations Research Society of Japan*, **57**, pp. 45–61, 2014.

[7] Y. Hu, H. Hashimoto, S. Imahori and M. Yagiura, "Efficient implementations of construction heuristics for the rectilinear block packing problem," *Computers & Operations Research*, **53**, pp. 206–222, 2015.

[8] 川島大貴，田中勇真，今堀慎治，柳浦睦憲，"3 次元箱詰め問題に対する構築型解法の効率的実現法，" 第 9 回情報科学技術フォーラム (FIT 2010) 講演論文集（第 1 分冊），pp. 31–38, 2010.

[9] 川島大貴，田中勇真，今堀慎治，柳浦睦憲，"3 次元パッキング問題に対する best-fit 法の効率的実現法，" 第 10 回情報科学技術フォーラム (FIT 2011) 講演論文集（第 1 分冊），pp. 29–36, 2011.

Copyright © by ORSJ. Unauthorized reproduction of this article is prohibited.