Vol.2013-AL-144 No.16
2013/5/18

3次元箱詰め問題に対する構築型解法の効率的実現法

田中 勇真[1,a)]　川島 大貴[2]　今堀 慎治[2,b)]　柳浦 睦憲[2,c)]

概要：3 次元箱詰め問題に対する代表的な構築型解法として，deepest-bottom-left (DBL) 法と 3 次元における best-fit (3BF) 法と呼ばれる 2 つの手法がある．本研究では，これらの構築型解法に対して，既存の手法と比べて理論計算量の少ない効率的な実現法を提案する．また，アルゴリズムの不要な探索を省略することで実計算時間を減らす工夫を加える．とくに，3BF 法では，この目的を実現するために分枝限定法を活用する．このような工夫を加えた結果，大規模な問題例においても実用的な時間で解を得られることを計算実験により確認した．

1. はじめに

配置問題とは，いくつかの対象物を互いに重ならないように与えられた領域内に配置する問題であり，多くの分野に応用を持つ代表的な生産計画問題の 1 つである．この問題は，対象物や領域の次元，形状，配置制約，目的関数等により非常に多くの種類があることが知られている [8], [21].

2 次元における配置問題には，2 次元平面上に複数の長方形を配置する長方形配置問題や，複数の多角形を配置する多角形配置問題などがあり，幾何学や組合せ最適化の分野で古くから研究されてきた．長方形配置問題は様々な種類の問題を含み，その多くは NP 困難である [17]．実用的な規模の問題例に対して厳密な最適解を求めることは難しいため，様々な近似解法が提案されてきた．その代表的なものに bottom-left (BL) 法 [2] と best-fit 法 [5] がある．

BL 法とは，初めに長方形を詰め込む順序を定め，なるべく下，同じ高さであればできる限り左に詰め込むことを繰り返す解法である．この解法を単純に実装すると長方形数 n に対し $O(n^4)$ 時間を要するが，データ構造を工夫することで $O(n^2 \log n)$ 時間 [10], [12] あるいは $O(n^2)$ 時間 [7] で実行できることが知られている．幅の定まった容器にすべての長方形を回転を許さず配置するとき，BL 法による解の精度は，長方形をその幅の降順に詰め込む場合に最適解の 3 倍以内に収まることが理論的に示されている [2]．この手法によって得られる解の精度は，長方形を詰め込む順序に依存するが，面積の大きい順等の簡単な基準でも比較的よい性能が得られる．また，メタ戦略を用いて良い順序を探索することで，より精度の高い解を見つける試みも行われている [11]．各反復において，長方形を置く際に，その配置場所を必ずしも最も下のできる限り左の位置に限定せず，左にも下にも動けない位置の 1 つに配置する BL 法に近いアルゴリズムも存在する [14], [18].

best-fit 法とは，注目している隙間を最大限に利用できる長方形を詰め込むことを繰り返す解法である．この解法はデータ構造の工夫により $O(n \log n)$ 時間 [13] で実行できることが知られている．また，best-fit 法で生成した配置の上部を一度取り出し，メタ戦略を用いて詰め込み直すことで，より精度の高い解を見つける試みも行われている [6].

3 次元箱詰め問題とは，直方体の容器に幅，高さ，奥行きを持つ複数の直方体を詰め込む問題の総称であり，様々な種類の問題を含んでいる．代表的なものに，コンテナ積み付け問題，3 次元ビンパッキング問題，3 次元ナップサックパッキング問題，3 次元ストリップパッキング問題などがある．また，3 次元箱詰め問題の応用として，コンテナへの荷物積み付けなどがある．

3 次元箱詰め問題の解法として様々な方法が提案されている．Bortfeldt と Gehring [3] はコンテナ積み付け問題に対して，容器を複数の層に分け，各層に対して箱詰めを行い，この層を重ねる解法を提案している．Lodi ら [19] は 3 次元ビンパッキング問題に対して，直方体を詰めた複数の層を作成し，各容器に層を詰め込む解法を提案している．Bortfeldt と Mack [4] は 3 次元ストリップパッキング問題に対して，コンテナ積み付け問題の解法を応用して，直方体を詰めた複数の層を作成し，容器に層を詰め込む解法を提案している．

本稿では，3 次元箱詰め問題に対して，2 次元箱詰め問題

[1] 成蹊大学，Seikei University
[2] 名古屋大学，Nagoya University
[a)] ytanaka@st.seikei.ac.jp
[b)] imahori@na.cse.nagoya-u.ac.jp
[c)] yagiura@nagoya-u.jp

を解くのに用いられる BL 法と best-fit 法を 3 次元へ拡張した解法の効率的実現法を提案する．2 次元の BL 法は自然に 3 次元へと拡張でき，直方体を詰め込む順序に従い，なるべく奥，同じ奥行きであればできる限り下，さらに同じ高さであればできる限り左に詰め込むことを繰り返す方法となる．本稿ではこれを deepest-bottom-left (DBL) 法と呼ぶ．この解法を単純に実装すると，直方体 n 個に対して $O(n^5)$ 時間を要するが，$O(n^4)$ 時間 [20] で実行できるアルゴリズムが存在する．一方，Allen ら [1] は，3 次元ストリップパッキング問題に対して，2 次元ストリップパッキング問題を解くのに用いられる best-fit 法を 3 次元に拡張した解法を提案している．本稿ではこれを 3 次元における best-fit (3BF) 法と呼ぶ．3BF 法は，直方体を 1 つ 1 つ配置していく構築型の解法であり，各反復において，最も奥，同じ奥行きであれば最も下，さらに同じ高さであれば最も左に詰め込むことのできる直方体の中から優先度の高いものを選んでその位置に配置する操作を繰り返す方法である．この解法を単純に実装すると，直方体数 n に対して $O(n^5)$ 時間を要する．また，この解法の効率的な実装法とその計算量を議論した論文は著者の知る限り存在しない．

本稿では，直方体数 n に対して $O(n^3 \log n)$ 時間で DBL 法を実行する効率的なアルゴリズムを提案する．また，t 種類 ($t \leq n$) のいずれかを形状として持つ合計 n 個の直方体に対して適用したときに $O(tn^2 \log n)$ 時間で 3BF 法を実行する効率的なアルゴリズムを提案する．これらのアルゴリズムの不要な探索を省略することによりさらなる効率化を行い，その効果を計算実験によって確認する．とくに，3BF 法に対しては，効率化手法として分枝限定法を用いており，次に置くべき候補を探す計算を省略する工夫を加えてある．このような工夫を加えた結果，直方体数 $n = 100,000$ 程度の大規模な問題例においても実用的な時間で解（直方体の配置）を構築できることを確認した．

なお，本稿で提案する手法は，コンテナ積み付け問題，3 次元ビンパッキング問題などにも容易に拡張でき，直方体の回転を許す問題例も扱うことができるが，以下ではとくにことわらない限り，回転を許さない 3 次元ストリップパッキング問題を対象とする．

2. 3 次元ストリップパッキング問題

3 次元ストリップパッキング問題の入力は，直方体の容器の幅 W と高さ H 及び直方体集合 $I = \{1, 2, \ldots, n\}$ に含まれる各直方体 $i \in I$ の幅 w_i，高さ h_i，奥行き d_i である．各直方体は $t\,(\leq n)$ 種類の形状集合 $K = \{1, 2, \ldots, t\}$ のいずれかを形状として持つものとする (すなわち直方体 i と j の形状がともに $k \in K$ であれば，$w_i = w_j$, $h_i = h_j$ かつ $d_i = d_j$ が成り立つ)．問題の目的は，各直方体を重ならないように容器に詰め込み，容器の可変長の奥行き D を最小化することである．これは，充填率 $VU(\%) = 100 \times \sum_{i \in I} w_i h_i d_i / WHD$ の最大化と言い換えることもできる．本稿では，座標の X 軸，Y 軸，Z 軸はそれぞれ直方体又は容器の幅，高さ，奥行き方向に対応し，右（同様に上，手前）に行くほど X（同様に Y，Z）座標は大きくなるものとする．直方体 i を配置したとき，i の最も奥かつ下かつ左の頂点の座標を (x_i, y_i, z_i) と表し，これを単に直方体 i の座標（あるいは参照点の座標）と呼ぶ．直方体の回転を許さないため，各直方体の配置を座標を用いて一意に定めることができる．この問題を定式化すると以下のようになる:

$$
\begin{aligned}
&\text{目的関数} \quad D \rightarrow \text{最小化} \\
&\text{制約条件} \quad 0 \leq x_i \leq W - w_i, \quad \forall i \in I, && (1)\\
&\qquad\qquad 0 \leq y_i \leq H - h_i, \quad \forall i \in I, && (2)\\
&\qquad\qquad 0 \leq z_i \leq D - d_i, \quad \forall i \in I, && (3)\\
&\qquad\qquad \text{任意の相異なる直方体対} && (4)\\
&\qquad\qquad i, j \in I \text{ が次の 6 つの不等式の}\\
&\qquad\qquad \text{少なくとも 1 つを満たす:}\\
&\qquad\qquad\quad x_i + w_i \leq x_j, \quad x_j + w_j \leq x_i,\\
&\qquad\qquad\quad y_i + h_i \leq y_j, \quad y_j + h_j \leq y_i,\\
&\qquad\qquad\quad z_i + d_i \leq z_j, \quad z_j + d_j \leq z_i.
\end{aligned}
$$

制約条件 (1)–(3) は各直方体 $i \in I$ が容器の中に配置されていることを，制約条件 (4) は直方体が互いに重ならないことを表す．

3. 準備

本研究では，文献 [12] で提案されている 2 次元上の BL 点を発見するアルゴリズム Find2D-BL を用いて，DBL 法と 3BF 法を効率的に行うアルゴリズムを提案する．そのため，本節では，Find2D-BL を紹介する．また，準備として 3 次元における BL 点 (DBL 点と呼ぶ) を定義する．3.1 節で一般的に多角形の重なり判定に用いられる no-fit polygon の説明を行い，3.2 節で Find2D-BL の説明を行う．3.3 節では BL 点と DBL 点について述べる．

3.1 多角形の重なり判定

no-fit polygon (NFP) 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 が重なりを持つような Q の参照点の座標の集合を P に対する Q の NFP と呼び，NFP(P,Q) と表す．P と Q がともに長方形の場合，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 の座標（本稿では左下の頂点の座標）の軌跡の内部領域である．同様の考え方を 3 次元に拡張することにより，直方体の重なりを判定する NFP を作成することができる．具体的には，位置 (x_j, y_j, z_j) に配置されている

直方体 j に対する直方体 i の NFP は次式で定義される．

$$NFP(j,i) = \{(x,y,z) \mid x_j - w_i < x < x_j + w_j, \\ y_j - h_i < y < y_j + h_j, \\ z_j - d_i < z < z_j + d_j\}.$$

以後，3 次元に拡張した NFP のことも単に NFP と呼ぶ．

3.2 2 次元平面上の BL 点発見法

既配置の長方形がいくつかあり，新たに長方形 i を配置しようとするとき，2 次元平面上の BL 点とは，長方形 i を既配置の長方形に重なりなく配置できる位置の中で，Y 座標が最も小さく，Y 座標が同じときは X 座標が最も小さい位置である．Find2D-BL は，NFP を用いて 2 次元平面上の BL 点を発見するアルゴリズムである [12]．なお，このアルゴリズムは，既配置の長方形同士の重複や，容器からのはみ出しがあっても正常に動作する．以下では Find2D-BL の考え方の概要と，BL 点の計算に要する時間を紹介する．

既配置の長方形全てに対し長方形 i の NFP を作成すると，i を配置したときの座標がそれら 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 をそこ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う場合がある．そこで，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．（これを特に inner-fit rectangle と呼ぶこともある [9].）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．

Find2D-BL は，走査線 (sweep-line) 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 の重複の数が分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，その走査線上の重複数が 0 である点の中で X 座標の値が最も小さい点が BL 点となる．図 1 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)，及び BL 点 (中央の黒丸) を示している．

既配置の長方形の数を m とすると，Find2D-BL は $O(m \log m)$ 時間で BL 点を見つけることができる．

3.3 3 次元における BL 点

本節では，本稿で使用する用語を定義する．直方体の 6 面の中で X-Y 平面に水平な 2 つの面のうち，Z 座標の大きい方をその直方体の前面，もう一方を背面と呼ぶ．一般性を失うことなく，Z 座標が 0 の X-Y 平面が容器の最も奥であるとし，これを容器の背面と呼ぶ．

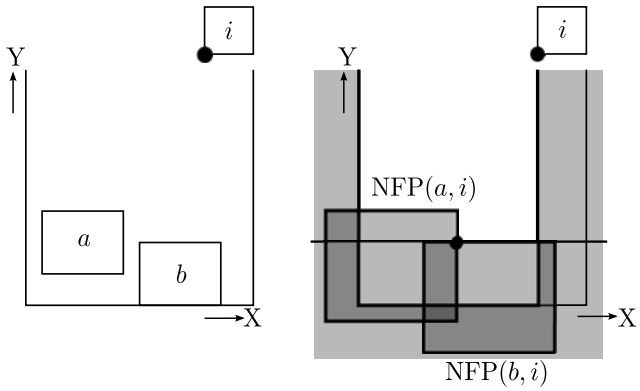

図 1 2 次元の BL 点の例

2 次元上の 2 点 $P_i = (x_i, y_i)$ と $P_j = (x_j, y_j)$ に対して，

$$(y_i < y_j) \vee (y_i = y_j \wedge x_i \leq x_j)$$

が成り立つとき，$P_i \preceq_{\mathrm{BL}} P_j$ と記すことにする．この順序関係を BL 順序と呼ぶ．既配置の長方形がいくつかあり，新たに長方形 i を配置するとき，BL 点は，容器からはみ出さずに長方形 i を既配置の長方形に重なりなく配置可能な位置の中で，BL 順序の意味で最も小さい位置となる．

次に 3 次元における BL 点を定義する．任意の 2 点 $P_i = (x_i, y_i, z_i)$ と $P_j = (x_j, y_j, z_j)$ に対して，

$$(z_i < z_j) \vee (z_i = z_j \wedge y_i < y_j) \\ \vee (z_i = z_j \wedge y_i = y_j \wedge x_i \leq x_j)$$

が成り立つとき，$P_i \preceq_{\mathrm{DBL}} P_j$ と記すことにする. この順序関係を DBL 順序と呼ぶ．既配置の直方体がいくつかあり，新たに直方体 i を配置しようとするとき，3 次元における BL 点とは，容器からはみ出さずに直方体 i を既配置の直方体に重なりなく配置可能な位置の中で，DBL 順序の意味で最も小さい位置である．以後，3 次元における BL 点のことを，DBL 点と呼ぶことにする．

4. deepest-bottom-left 法

本節では，deepest-bottom-left (DBL) 法を効率的に実現するアルゴリズムを提案する．なお，計算量の算定と実用的な高速化手法については，本稿では紙数の都合上説明を省略する．詳細は文献 [15] を参照されたい．

ある直方体 i を DBL 点に配置するとき，i の背面は他の直方体の前面または容器の背面と接しているはずである．よって，既配置の直方体の前面と容器の背面の座標の値に z_i の候補を絞って DBL 点を探索すれば良いことが分かる．z_i の候補となる Z 座標の 1 つを z'（例えば直方体 j の前面であれば $z' = z_j + d_j$ ）とする．このとき，Z

座標の値が z' より大きく $z'+d_i$ より小さい位置全てからなる層 (すなわち集合$\{(x,y,z) \in R^3 \mid z' < z < z'+d_i\}$) と共通部分を持つ既配置の直方体全てを X-Y 平面に射影する．この X-Y 平面上において，直方体 i の背面が既配置の直方体の前面あるいは容器の背面と接する領域の中で 2 次元の BL 点を探索する．この探索で見つかった BL 点は DBL 点の候補となる．

実際に DBL 点を得るためには，既配置の直方体の前面と容器の背面を面の座標 (左下の頂点の座標) の DBL 順序の昇順に並び替え，この順に従って各面と直方体 i の背面が接する領域上で 2 次元の BL 点を探索する．初めて BL 点を見つけたとき，探索中の面と Z 座標が同じ面が探索の対象となっている X-Y 平面上にないときは，見つけた BL 点が i の DBL 点となる．一方，初めて BL 点を見つけた面と同じ Z 座標を持つ面が複数ある場合は，最初に見つけた BL 点が DBL 点になるとは限らない．この場合は，同じ Z 座標を持つ各面上の 2 次元の BL 点の中で，DBL 順序の意味で最も小さい座標が DBL 点となる．ただし，どこに配置したとしても，現在発見されている最良の BL 点よりも DBL 順序の意味で大きい位置にしか配置できないと結論付けられる前面を探索する必要はない．

上記の方法を実際に実装する場合は，まず既配置の直方体 j のおのおのに対して次に配置する直方体 i の 3 次元の NFP，NFP(j,i) を作成する．ある既配置の直方体 j の前面に接するように直方体 i を置くためには，i の座標が NFP(j,i) の前面領域の内部になければならない (i の座標が NFP(j,i) の境界線上にあるときは，i の背面と j の前面は境界のみが接するため BL 点の候補にならないことに注意)．NFP(j,i) の前面領域の内部で，i が他のどの直方体とも重複を持たないための条件は i の座標がどの NFP の内部にも含まれないことである．この状況は平面 $z = z_j + d_j$ で切った 2 次元平面で考えれば良い．

直方体 j に対する i の NFP の前面領域と重なる既配置の直方体の NFP の数を k とすると，Find2D-BL を使用することにより，$O(k \log k)$ 時間で NFP(j,i) の前面の BL 点を見つけることができる．直方体 j の候補として既配置の直方体を DBL 順序に従って調べていき，そのおのおのに対して以上の操作を行うことで DBL 点が求まる．これを与えられた n 個の直方体を全て配置するまで繰り返すことにより，提案した DBL 法は $O(n^3 \log n)$ 時間で実現できる．

5. 3 次元における best-fit 法

本節では，3 次元における best-fit (3BF) 法を効率的に実現するアルゴリズムを提案する．5.1 節で 3BF 法およびその $O(tn^2 \log n)$ 時間での実現法について説明する．5.2 節では，さらなる高速化のためのアイデアを述べる．

5.1 3 次元における best-fit 法の効率的実現法

Find2D-BL を用いた 3BF 法の効率的実現法を提案する．3BF 法を説明する前に，まず 2 次元における best-fit 法を説明する [5]．2 次元における best-fit 法は長方形を一つずつ配置する構築型解法である．まず，長方形間に幅の降順などの優先順位を定めておく (その他の順位付けルールについては文献 [5] を参照)．既配置の長方形 j の上辺 (Y 座標が $y_j + h_j$ の辺) と容器の底辺を Y 座標の昇順に並べ，この順に従って各辺上で以下のことを行う．現在注目している辺から X 軸に平行に両側に線を伸ばし，容器の壁または既配置の長方形に遮られた位置までの線分を考える．この線分を，lowest available gap を略して LAG と呼ぶ．LAG 上に配置可能な未配置の長方形の中から優先順位の最も高いものを選び，LAG 上の X 座標の最も小さい位置に長方形の左下の頂点を配置する (文献 [5] では配置位置について他の方法も提案されている)．LAG から配置した長方形の底辺と重なる部分を削除した残りの線分を新たな LAG として同様の操作を繰り返す．LAG 上に未配置のいずれの長方形も置けなくなったら，次の辺に移動し同様の操作を繰り返す．このように LAG の Y 座標はアルゴリズムの実行全体を通して単調非減少であり，新たな長方形 i を配置するとき，既配置の長方形はそれ以前の時点での LAG 上にその底辺が置かれている．その結果，i を配置する時点では，配置済みの長方形でその底辺が現在の LAG より上にあるものは存在しない．したがって，新たな長方形 i を LAG 上に配置する際，i の幅が LAG の長さ以下であれば，i の高さにかかわらず既配置のどの長方形とも重複しない．

上述の 2 次元における best-fit 法の実現法では，単純に 3 次元に拡張することはできない．そこで，best-fit 法の異なる解釈を考える．未配置の長方形全てに対して BL 点を求めたと仮定する．その中で BL 順序の意味で最も小さい位置は，上述のアルゴリズムが注目する LAG 上の X 座標の最も小さい位置に相当する．よって，2 次元における best-fit 法は以下のように言い換えられる．まず，未配置の長方形全てに対して BL 点を求める．次に，その中で BL 順序の意味で最も小さい位置に，この位置を BL 点として持つ長方形の中から優先順位の最も高いものを配置する．この操作を未配置の長方形がなくなるまで繰り返す．このルールは自然に 3 次元に拡張することができ，その結果 3 次元における best-fit 法が得られる．

3BF 法は，全ての未配置の直方体に対する DBL 点の中で DBL 順序の意味で最も小さい位置に，配置可能な直方体の中で最も優先順位が高いものを配置する操作を，未配置の直方体がなくなるまで繰り返す方法である．そのような配置位置およびそこに配置できる直方体集合は，たとえば未配置の直方体全てに対して DBL 点を計算することで得られるが，以下ではこの計算を高速に行う方法を提案する．

前述のように，ある直方体 i を DBL 点に配置するとき，

i の背面は他の直方体の前面または容器の背面と接する．よって，既配置の直方体の前面と容器の背面の Z 座標の値に z_i の候補を絞って探索すればよいことが分かる．

これらの候補の各 Z 座標 z' における X-Y 平面を z' の昇順に並べ，この順に従って以下の操作を行う．現在注目している Z 座標 z' における X-Y 平面と重複をもつ既配置の直方体をこの平面で切断したときの切断面を既配置の長方形とする．なお，このとき，直方体の前面（背面）の Z 軸の値が z' と等しい場合は，この平面との重複を持たない（持つ）ものとする．次に，全ての未配置の直方体の背面に対して，それを新たに置く長方形として 2 次元の BL 点を探索する．このとき，1 つ以上の未配置の直方体に対して BL 点が見つかった場合は，BL 順序の意味で最も小さい BL 点は，DBL 順序の意味においても最も小さい．したがって，この BL 点に配置可能な未配置の直方体の中で最も優先順位の高いものを配置する．（全ての既配置の直方体の背面の Z 座標は z' 以下であるため，2 次元平面上に重なりなく置くことが出来れば 3 次元空間上でも重複のないことが保証できる．）そして，次の直方体の配置位置を探索するとき，同じ面に他の直方体が配置できる可能性があるので，探索を同じ面から始める．一方，全ての未配置の直方体に対して BL 点が見つからなかった場合は，現在の面に配置可能な直方体は存在しないため，次の面の探索に移行する．これを未配置の直方体がなくなるまで繰り返すことにより，3BF 法を実現できる．

直方体の形状が t 種類あるような合計 n 個の直方体が与えられたとき ($t \leq n$)，これらを上記の方法で配置するのに要する計算時間を議論する．いくつかの直方体が配置されているとき，未配置の直方体の背面のおのおのに対して現在の X-Y 平面上で Find2D-BL を用いて BL 点を探索する．ただし，背面の形状が同じものは BL 点が同じになるので，その中の 1 つについて Find2D-BL を呼び出せばよく，背面の形状の数は直方体の形状の種類数以下なので，Find2D-BL の呼び出し回数は $O(t)$ 回である．したがって，以上の操作にかかる計算時間は $O(tn \log n)$ である．また，配置可能なものが 1 つ以上見つかったとき，それらの中から優先順位の最も高いものを選ぶ操作は $O(n)$ 時間で可能である．以上の計算を行う度に，1 つの直方体の配置が決定するか，あるいは探索中の X-Y 平面に配置できる未配置の直方体が存在しないことが結論できる．そのいずれも高々 n 回しか起こり得ない．よって，t 種類，n 個の直方体の配置は $O(tn^2 \log n)$ の計算時間で可能である．

5.2 アルゴリズムの実用的高速化

本節では 5.1 節で提案したアルゴリズムに対する，実用的な高速化のための 3 つのアイデアを提案する．各アイデアの詳細については文献 [15] を参照されたい．

探索範囲の削減

直方体の配置位置は既配置の直方体の NFP の前面領域上である．この性質を使って，z_i の候補を絞ることを前節で議論した．さらに，Find2D-BL が探索の対象とする領域を，X-Y 平面全体から注目している直方体の NFP の前面領域に絞り，その範囲内に新たな直方体を置く際に重複を持ち得るものに既配置の直方体を限定して計算を行うことにより，計算時間の向上が見込まれる．探索は，直方体の NFP の前面の座標（左下の座標）の DBL 順序の昇順に行う．これにより，ある未配置の直方体に対して 1 つ BL 点が求まっていると，それより DBL 順序の大きい点を見つける必要がないため，次の NFP の前面の座標がその BL 点よりも DBL 順序の意味で大きい場合は，それ以降の全ての NFP の前面に対する Find2D-BL の計算を省略できる．また，Find2D-BL の計算は DBL 順序の昇順に候補を調べて行くため，既に見つかっている BL 点の位置を超えた時点で，残りの計算を省略することができる．

X-Y 平面全体を調べずに NFP の前面領域に探索を絞る上述の方法は，探索中の X-Y 平面に前面がある既配置の直方体の数が少ないときに特に効果が期待できる．ただし，既配置の直方体の配置や次に配置するものの形状によっては，計算対象とする既配置の直方体の数があまり減らない場合もある．また，探索中の X-Y 平面に前面がある直方体が多数ある場合は，その多くに対して Find2D-BL を呼び出す必要が生じ，その結果 X-Y 平面全体に対して 1 度だけ Find2D-BL を呼び出す場合よりも計算量が増えてしまう可能性がある．実際，このアイデアを単純に実装してしまうと，全ての直方体の配置に $\Omega(tn^3 \log n)$ 時間かかってしまう例を容易に作成できる．このため，提案手法では，5.1 節で説明した方法と上述の方法を組み合わせることで，最悪計算量の増加を防ぐ仕組みを組み込んでいる．

5.1 節で提案したアルゴリズムは，1 つの直方体を配置するために，探索中の X-Y 平面上で未配置の直方体全てに対して Find2D-BL を用いて BL 点の探索を行う．この計算の効率化を図るため，

- Find2D-BL の呼び出し回数を減らす，
- Find2D-BL の探索範囲を減らす，

ことを考える．直方体集合 $I = \{1, 2, \ldots, n\}$ に対し，未配置の直方体集合を $I_{\mathrm{u}} \subseteq I$ とする．$w_{\min} = \min_{j \in I_{\mathrm{u}}} w_j$, $h_{\min} = \min_{j \in I_{\mathrm{u}}} h_j$, $d_{\min} = 1$ と定義して $w_{\min}, h_{\min}, d_{\min}$ を幅，高さ，奥行きとして持つ直方体 r を考える (奥行きは BL 点の探索に影響を与えない)．これを最小直方体と呼ぶこととする．r の X-Y 平面への射影は未配置の直方体のいずれの射影よりも小さいか等しい．探索中の X-Y 平面上の r の BL 点をBL_rとすると，BL_rは未配置の直方体のいずれの BL 点よりも DBL 順序の意味で小さいか等しい．すなわち，BL_rは全ての未配置の直方体の DBL 点に対する下界となる．もし，BL_rと等しい BL 点をもつ未

配置の直方体が存在するならば，その直方体より優先順位の低い直方体については探索する必要がない．また，探索中の X-Y 平面上にBL_rが存在しなければ，その X-Y 平面は探索する必要がないことがすぐに分かる．さらに，BL_rより DBL 順序の意味で小さいか等しい領域は Find2D-BL により探索する必要がないことも分かる．詳細は省略するが，このような事実から提案手法では，Find2D-BL の呼出し回数および探索範囲の削減を実現している．

指定点への配置可能性確認による高速化

前項で述べたように，BL_rと等しい BL 点をもつ未配置の直方体が存在するならば，その直方体より優先順位の低い直方体については探索する必要がない．また，たとえ優先順位がそのような直方体よりも高いものが存在したとしても，BL 点がBL_rでなければ配置する候補から除外することができる．すなわち，BL_rと等しい BL 点をもつ未配置の直方体の中で最も優先順位の高いものを見つけることができれば，配置する直方体が決定する．ここでは，BL_rと BL 点が一致する最も優先順位の高い未配置の直方体を見つける探索を効率よく行う手法を提案する．

未配置の直方体が探索中の X-Y 平面上のある位置 q に配置可能か否かを確かめるには，未配置の直方体を q に配置したときに全ての既配置の直方体と重複を持たないことと，容器内に配置されていることを確認すれば良い．この確認は未配置の直方体の各形状 (t 種類以下) に対して既配置の直方体数 m (n 個以下) の線形時間で可能なため，$O(tm)$ 時間で実行可能である．この事実より次のような高速化が可能である．BL_rが決定したときに，BL_rに対して未配置の直方体が配置可能か否かの確認を優先順位の高い順に行う．配置可能な直方体を初めて見つけたとき，その直方体をBL_rに配置する．こうすることで，BL_rに配置可能な直方体が存在する場合に限り，$O(m \log m + tm) = O(n \log n + tn)$ 時間（BL_rを計算する時間と未配置の直方体のおのおのについてBL_rに置けるか否かを確認する時間）で 1 つの直方体を配置でき，全ての未配置の直方体に対し BL 点を探索する場合の計算量 $O(tm \log m) = O(tn \log n)$ に比べ大幅な改善となる．ただし，BL_rに配置可能な直方体がない場合は，この方法では次の直方体の配置位置を決定できないため，この確認作業は計算時間を増やすだけある．しかし，実験によりBL_rに配置可能な直方体が存在する場合が非常に多いことが分かっており，この高速化の効果により全体の計算時間の削減が期待できる．

次に未配置の直方体を位置 q に配置したときに，重複がないかどうか確認すべき既配置の直方体の候補を絞る 3 つの方法を述べる． q は探索中の X-Y 平面上に存在する．これより，この平面の Z 座標 z' より前面の Z 座標が小さい既配置の直方体と， q に配置した直方体が重複をもつことはない．よって，注目している値 z' における X-Y 平面と重複をもつ既配置の直方体と， q に配置した直方体が重複がないことを確認すればよいことがわかる．

$w_{\max} = \max_{j \in I_{\mathrm{u}}} w_j$， $h_{\max} = \max_{j \in I_{\mathrm{u}}} h_j$, $d_{\max} = 1$ と定義して $w_{\max}, h_{\max}, d_{\max}$ を幅，高さ，奥行きとして持つ直方体 $r_{\max}$ を考える（奥行きは直方体の重複に影響を与えない）． $r_{\max}$ の X-Y 平面への射影は未配置の直方体のいずれの射影よりも大きいか等しい．よって， $r_{\max}$ を位置 q に配置したときに $r_{\max}$ と重複しない既配置の直方体は， q に配置したいずれの未配置の直方体とも重複を持たない．これより，位置 q に $r_{\max}$ を配置したときに重複を持つ既配置の直方体のみに候補を絞ることができる． $r_{\max}$ を q に配置したときに既配置の直方体と重複を持つか否かを確認するため，この絞込みには $O(m)$ 時間を要する．

上記の 2 つの方法により重複を持つか否かを確認すべき候補と判断されたある既配置の直方体 j について考える．この直方体 j は $(x_q < x_j + w_j) \wedge (y_q < y_j + h_j)$ を満たす．同じく上記の 2 つの方法により確認すべき候補と判断され，$(x_k \leq x_j) \wedge (y_k \leq y_j)$ を満たす既配置の直方体 k があるとする．位置 q に配置した未配置の直方体 i が $(x_j < x_q + w_i) \wedge (y_j < y_q + h_i)$ を満たすとき， i と j は重複を持つ．このとき，$(x_k < x_q + w_i) \wedge (y_k < y_q + h_i)$ も成り立つので， i と k も重複を持つ．このことより，直方体 j に対して $(x_k \leq x_j) \wedge (y_k \leq y_j)$ を満たす直方体 k が存在するならば， j に対して重複の有無を確認する必要はない．これにより，重複の有無を確認すべき候補を減らすことができる．具体的には，まず上記の 2 つの方法により重複の有無を確認すべき候補を X-Y 平面に射影したときの左下の頂点の座標の BL 順序の昇順に並び替えたリストを N_{BL}とする． N_{BL}の s 番目の直方体が j であることを $N_{\mathrm{BL}}(s) = j$ と表す． N_{BL}の $(s-1)$ 番目までの直方体の中で最も X 座標が小さい直方体の X 座標を x^* とする．N_{BL}の s 番目の直方体 $j = N_{\mathrm{BL}}(s)$ と， N_{BL}の $(s-1)$ 番目までの各直方体 k に対して $y_k \leq y_j$ が成り立つ．よって，$x^* \leq x_j$ を満たすならば，直方体 j は重複の有無を確認する必要はない．そうでなければ，直方体 j は重複の有無を確認すべき候補となる．これをリスト N_{BL}の順序に従って行っていくことで，確認すべき候補を絞ることができる．候補を BL 順序の昇順に並び替えることに $O(m \log m)$ 時間，それ以外の作業に $O(m)$ 時間を要する．

以上の高速化のアイデアを組み込んだ結果，未配置の直方体がある位置 q に重複なく配置可能か否かの確認は $O(m \log m + tm)$ 時間を要する．単純に確認するときの時間 $O(tm)$ に比べ，計算量が増えているが，確認すべき直方体数を大幅に削減できるため，実際の計算時間は削減されている．なお，BL_rの決定に $O(m \log m)$ 時間を要するため，位置BL_rに重複なく配置可能か否かの確認は $O(m \log m + tm)$ 時間で実行でき，計算量は単純に確認する場合と等しくなるが，確認作業の高速化により実計算時間の削減が期待できる．

分枝限定法

ここまでに提案した高速化手法により，多くの場合において，1 つの直方体を配置するために，全ての未配置の直方体に対して BL 点の探索を行う必要はなくなった．しかし，BL_rに配置できる直方体が存在しない場合は探索の省略の効果は小さい．本節では，分枝限定法を用いてBL_rに配置できる直方体が存在しない場合においても探索を削減できるアルゴリズムを提案する．

未配置の直方体が残っている形状を優先順位の順に並べた順列を σ とし，σ の u 番目から v 番目までの形状の集合を $\sigma(u,v)$ と表す．本研究で用いる分枝限定法の分枝操作は，順列 σ を半分に分割することで実現する．探索中の X-Y 平面において，各部分問題に対して直方体の BL 点の上下界値を求め，その部分問題の最適解が得られるか，またはその部分問題の最適解が元の問題の最適解とならないことが分かれば，その下の節点の探索を省略する．

下界値の計算のため，直方体 $r(u,v)$ を，その X-Y 平面への射影が $\sigma(u,v)$ に含まれる直方体のいずれの射影よりも小さいか等しい直方体とする．このとき, $r(u,v)$ の BL 点は $\sigma(u,v)$ に含まれる直方体のいずれの BL 点よりも Y 座標が小さいか，Y 座標が等しく X 座標が小さいか等しい．したがって，この BL 点の座標は現在の部分問題の下界として作用する．この下界値テストで終端できなかった部分問題に対しては，直方体 $r(u,v)$ の BL 点に対して $\sigma(u,v)$ に含まれる直方体が配置可能か否かの確認を優先順位の高い順に行う．配置可能な直方体 i を初めて見つけたとき，この位置へ直方体 i を配置することで，集合 $\sigma(u,v)$ に対応する部分問題の最適解が得られる．また，これは元の問題の上界を与える．

6. 計算実験

計算実験により提案手法の計算効率を評価する．なお，本稿では 3BF 法に関する実験結果のみを示す．DBL 法に関する実験結果については文献 [15] を参照のこと．まず，Find2D-BL を用いない単純な 3BF 法との計算時間の比較を行い，提案手法の有効性を確認する．また，実用的高速化を行わないアルゴリズム（5.1 節のアルゴリズム）との計算時間の比較によって，5.2 節で提案した実用的高速化の有効性を確認する．計算実験は全て Dell Precision 470 (Xeon 2.83GHz，6MB cache，24GB memory) 上で行い，実験には，文献 [16] で用いられている問題例を使用した．使用した問題例の特徴として，隙間のない配置が存在すること，偶然の一致を除いて全ての直方体が相異なる形であること（すなわち $t \approx n$）が挙げられる．

まず，単純な 3BF 法との比較を表 1 に示す．単純な 3BF 法とは，DBL 点の候補を DBL 順序の最も小さいものから順に，優先順位の順に各未配置の直方体を配置したときに既配置の直方体と重なりがないかを確認し，重なりがなければその位置に配置することを繰り返す方法である．ある位置に対して全ての未配置の直方体 (たかだか tk 種類) と既配置の直方体 m 個との重なりの確認は $O(tm)$ 時間で可能である．DBL 点の候補は $O(m^3)$ 個存在する．重なりの確認は各候補に対して，1 度ずつしか行わないため，n 個の直方体の詰め込みは $O(tn^4)$ の計算時間で可能である．このアルゴリズムを 3BF-Simple，本稿で提案した分枝限定法を用いたアルゴリズムを 3BF-BB と呼ぶ．また，3 次元における best-fit 法に用いる優先順位には幅の降順 を用いた．表 1 に結果を示した実験では，直方体の数 n が $50, 100, \ldots, 500$ の 10 通り，計 50 問を使用した．なお，両手法によって得られる解は等しく，表中の VU は両手法共通の充填率を表す．

n が大きくなるにつれて 3BF-BB の計算時間はほとんど変わらないが，3BF-Simple の計算時間は急速に増大していることが観測できる．$n = 500$ のとき，3BF-BB の計算時間は 0.01 秒であるのに対して，3BF-Simple の計算時間は 12 分程度かかっている．

次に，5.2 節で提案したアルゴリズムの高速化の効果を調べるため，実用的高速化を行わないアルゴリズム 3BF-NFP（5.1 節のアルゴリズム），探索範囲の削減のみ行うアルゴリズム 3BF-LB，指定点に対する確認の高速化を行ったアルゴリズム 3BF-OPC，分枝限定法を含む全ての実用的高速化を実装したアルゴリズム 3BF-BB の比較を表 2 に示す．この表に結果を示した実験では，直方体の数 n が $1000, 2000, \ldots, 10000$ と $50000, 100000$ の問題例，計 60 問を使用した．比較する 4 つの手法によって得られる解は等しく，表中の VU は共通の充填率を表す．制限時間を 3600 秒とし，これを超えた場合は実験を中止することとした．

表よりアルゴリズムの各高速化の効果は非常に大きいことが観測できる．$n = 10,000$ の問題例において，3BF-NFP は 30 分以上を要するのに対し，3BF-LB は 137 秒，3BF-OPC は 26 秒，3BF-BB は 4 秒程度で計算が可能である．すなわち，この規模の問題例に対して，3BF-BB は 3BF-NFP の約 500 倍速いことが分かる．また，$n = 100,000$ の大規模な問題例に対しても，3BF-BB の計算時間は 10 分以下と非常に速い時間で配置が行えている．

7. まとめ

2 次元の長方形配置問題に対する代表的な構築型解法である BL 法と best-fit 法は，3 次元箱詰め問題に自然に拡張できる．本研究では，これらを 3 次元に拡張した DBL 法と 3BF 法の効率的実現法を提案した．まず，DBL 法については，これを $O(n^3 \log n)$ の計算時間で実現するアルゴリズムを提案した．著者の知る限り，既存のアルゴリズムには $O(n^4)$ 時間のものしか存在せず大きな改善といえる．次に，3BF 法については，これを $O(tn^2 \log n)$ の計算時間で実現するアルゴリズムを提案した．また，実用的高速化

表 1 単純な 3BF 法との比較

		time (sec)	
n	VU (%)	3BF-Simple	3BF-BB
50	64.08	0.01	0.00
100	71.24	0.39	0.00
150	72.49	3.28	0.00
200	72.96	12.17	0.00
250	76.25	34.82	0.00
300	78.83	78.39	0.01
350	79.01	167.95	0.01
400	79.62	308.23	0.01
450	79.96	478.86	0.01
500	80.02	741.98	0.01

表 2 アルゴリズムの実用的高速化の効果

		time (sec)			
n	VU (%)	3BF-NFP	3BF-LB	3BF-OPC	3BF-BB
1000	78.23	16.05	1.34	0.51	0.05
2000	83.92	74.73	5.36	1.59	0.18
3000	86.92	171.54	11.75	3.43	0.39
4000	88.78	303.64	18.97	5.32	0.67
5000	90.42	606.15	37.99	9.52	1.06
6000	89.88	718.20	45.42	11.42	1.49
7000	91.22	1004.27	72.11	16.17	2.03
8000	91.62	1245.28	89.94	19.48	2.64
9000	92.21	2026.62	126.05	27.75	3.36
10000	92.61	2014.90	137.06	26.31	4.09
50000	94.00	-	-	1594.35	110.21
100000	94.95	-	-	-	560.17

のアイデアを複数提案し，これらを組み込んだ解法を実装した．計算実験を行ったところ，実用的高速化の効果により，直方体数が 10,000 の場合に計算時間を約 500 倍速くできることを計算実験によって確かめた．また，直方体数 100,000 の問題例に対しても 10 分以下と実用的な時間で解を構築できることを確認した．

本研究では直方体数最大 100,000 の問題例に対して計算実験を行ったが，直方体数が小さい問題例に対して得られた充填率は 8 割程度であり，まだ改善の余地がある．3BF 法は得られた配置の前半部分は充填率が高いが，後半部分は低いという特徴がある．今後は，提案手法で得られた解の後半部分を DBL 法を用いて改善していくなどの方法により，解の精度の向上を目指していきたい．

参考文献

[1] S. D. Allen, E. K. Burke, G. Kendall: A hybrid placement strategy for the three-dimensional strip packing problem, European Journal of Operational Research, 209 (2011), 219–227.
[2] B.S. Baker, E.G. Coffman Jr., R.L. Rivest: Orthogonal packing in two dimensions, SIAM Journal on Computing, 9 (1980), 846–855.
[3] A. Bortfeldt, H. Gehring: A hybrid genetic algorithm for the container loading problem, European Journal of Operational Research, 131 (2001), 143–161.
[4] A. Bortfeldt, D. Mack: A heuristic for the three-dimensional strip packing problem, European Journal of Operational Research, 183 (2007), 1267–1279.
[5] E. K. Burke, G. Kendall, G. Whitwell: A new placement heuristic for the orthogonal stock-cutting problem, Operations Research, 52 (2004), 655–671.
[6] E. K. Burke, G. Kendall, G. Whitwell: A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock-cutting problem, INFORMS Journal on Computing, 21 (2009), 505–516.
[7] B. Chazelle: The botom-left bin-packing heuristic: an efficient implementation, IEEE Transactions on Computers, C-32 (1983), 697–707.
[8] H. Dyckhoff: A typology of cutting and packing problems, European Journal of Operational Research, 44 (1990), 145–159.
[9] A. M. Gomes, J. F. Oliveira: A 2-exchange heuristic for nesting problems, European Journal of Operational Research, 141 (2002), 359–370.
[10] P. Healy, M. Creavin, A. Kuusik: An optimal algorithm for rectangle placement, Operations Research Letters, 24 (1999), 73–80.
[11] E. Hopper, B. C. H. Turton: A review of the application of meta-heuristic algorithms to 2D strip packing problems, Artificial Intelligence Review, 16 (2001), 257–300.
[12] S. Imahori, Y. Chien, Y. Tanaka, M. Yagiura: Enumerating bottom-left stable positions for rectangles with overlap, 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.25–30.
[13] S. Imahori, M. Yagiura: The best-fit heuristic for the rectangular strip packing problem: an efficient implementation and the worst-case approximation ratio, Computers & Operations Research, 37 (2010), 325–333.
[14] S. Jakobs: On genetic algorithms for the packing of polygons, European Journal of Operational Research, 88 (1996), 165–181.
[15] 川島 大貴: 3 次元箱詰め問題に対する 2 つの構築型解法の効率的実現法, 名古屋大学大学院情報科学研究科 修士論文，2012.
[16] 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.
[17] J. Leung, T. Tam, C. S. Wong, G. Young, F. Chin: Packing squares into square, Journal of Parallel and Distributed Computing, 10 (1990), 271–275.
[18] D. Liu, H. Teng: An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles, European Journal of Operational Research, 112 (1999), 413–420.
[19] A. Lodi, S. Martello, D. Vigo: Heuristic algorithms for the three-dimensional bin packing problem, European Journal of Operational Research, 141 (2002), 410–420.
[20] L. Wang, S. Guo, W. Zhu, A. Lin: Two natural heuristics for 3D packing with practical loading constraints, Proceedings of the 11th Pacific Rim International Conference on Trends in Artificial Intelligence (PRICAI), LNAI 6230, pp.256–267, 2010.
[21] G. Wäscher, H. Haußner, H. Schumann: An improved typology of cutting and packing problems, European Journal of Operational Research, 183 (2007), 1109–1130.