RA-005

3次元パッキング問題に対する best-fit 法の効率的実現法

An efficient implementation of the best-fit algorithm for the three-dimensional packing problem

川島大貴* 田中勇真* 今堀慎治† 柳浦睦憲*
Hiroki Kawashima Yuma Tanaka Shinji Imahori Mutsunori Yagiura

1 はじめに

切り出し・詰め込み問題とは, 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 [1, 2].

2 次元における詰め込み問題には, 2 次元平面上に複数の長方形を配置する長方形詰め込み問題や, 複数の多角形を配置する多角形詰め込み問題等があり, 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. 長方形詰め込み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 [3]. 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, 様々な近似解法がこれまでに提案されてきた. その代表的なものに best-fit 法 [4] がある. 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を工夫することで $O(n \log n)$ 時間 [5] で実行できることが知られている. また, best-fit 法で生成した解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, 幅, 高さ, 奥行きを持つ 1 つの容器（コンテナ）に直方体を詰め込み, 隙間を最小化するような直方体の配置を決めるコンテナ積み付け問題 (single container loading problem), 幅, 高さ, 奥行きを持つ容器が複数与えられ, 全ての直方体を詰め込むときに使用する容器の個数を最小化するような直方体の配置を求める 3 次元ビンパッキング問題 (3-dimensional bin packing problem), 幅, 高さ, 奥行きを持つ容器と価値が付加された複数の直方体が与えられ, 詰め込んだ直方体の価値の合計を最大化するような直方体の配置を決める 3 次元ナップサックパッキング問題 (3-dimensional knapsack packing problem), 幅, 高さと可変長の奥行きを持つ直方体の容器が与えられ, 与えられた全ての直方体を詰め込むときの容器の奥行きを最小化する 3 次元ストリップパッキング問題 (3-dimensional strip packing problem) 等がある. 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等がある.

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 [7] 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 [8] 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. Egeblad と Pisinger [9] は 3 次元ナップサックパッキング問題に対して, 直方体同士の相対関係を直方体番号の 3 つの順列により表し, 配置する解法を提案している. Bortfeldt と Mack [10] 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.

Allen ら [11] 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 次元の best-fit 法と呼ぶ (3 次元であることが明らかな場合は単に best-fit 法と呼ぶ). 3 次元の best-fit 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, 3 次元の best-fit 法の効率的実現法を提案する. 具体的には, 3 次元における best-fit 法を, 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 $O(tn^2 \log n)$ 時間で動作する効率的アルゴリズムを提案する. また, こ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

*名古屋大学大学院情報科学研究科
†名古屋大学大学院工学研究科

率化を図り, その効果を計算実験によって確認する. このような工夫を加えた結果, 直方体数 $n = 10000$ 程度の大規模な問題例においても実用的な時間で解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等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. 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:

目的関数　$D \rightarrow$ 最小化

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

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 アルゴリズム

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, 3 次元の箱詰めを行うアルゴリズムを提案する. 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 3 次元箱詰めの解法について説明する. 3.4 節ではアルゴリズムの高速化の方法について説明する.

3.1 no-fit polygon

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. 図 1 は NFP の例であり, 太線が NFP の境界を表す.

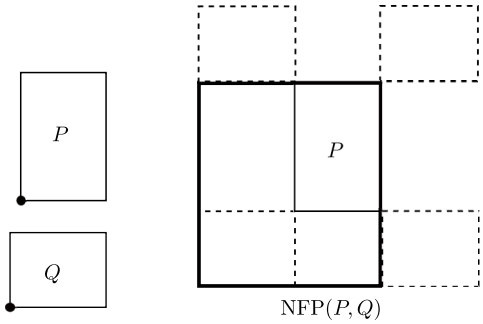

図 1: NFP の例

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,

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

である. 以後, 3 次元に拡張した NFP のことも単に NFP と呼ぶ.

3.2 2 次元上の BL 点発見法

既配置の長方形がいくつかあり, 新たにある長方形 i を配置しようとするとき, 2 次元上の 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 次元上の BL 点を発見するアルゴリズムである. なお, このアルゴリズムは, 既配置の長方形同士が重複していたり, 容器からはみ出している長方形があっても

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 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. （これを特に inner-fit rectangle (IFR) と呼ぶこともある [13].）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分かる.

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, 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 2 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.

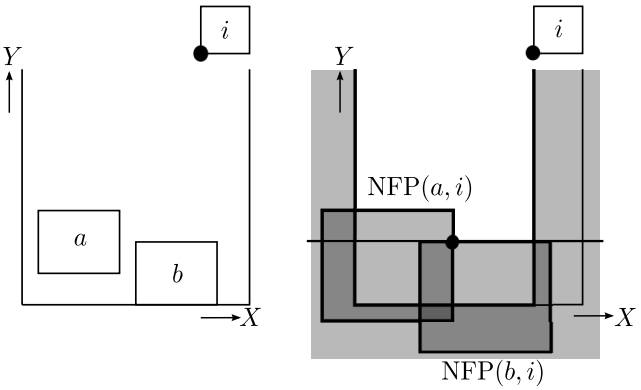

図 2: 2 次元の BL 点の例

3.3 3 次元における best-fit 法

本節では, 3 次元における BL 点の定義を説明した後, Find2D-BL を用いた 3 次元における best-fit 法のアルゴリズムを提案する. はじめにこれから使用する用語を定義する. 直方体の 6 面の中で X-Y 平面に平行な 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, 容器の背面と呼ぶ.

次に 3 次元上の BL 点について説明する. 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,

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

が成り立つ時, $P_i \preceq_{\mathrm{BL}} P_j$ 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 次元上の BL 点とは, 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, BL 順序の最も小さい位置である. 以後, 3 次元上の BL 点のことを, 3 次元における BL 点または単に BL 点と呼ぶことにする.

3 次元の best-fit 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 [4]. これ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 [4] 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 [4] 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.

さて, 以下では, best-fit 法の異なる解釈を考える. 前述の BL 順序は 2 次元においても同様に定義できる. 未配置の長方形全てに対して BL 点を求める. その中で BL 順序の最も小さい位置は, 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. 未配置の長方形全てに対して BL 点を求

める．その中で BL 順序の最も小さい位置に, 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．これ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．

3 次元における best-fit 法は, 未配置の直方体が 1 つ以上配置可能な位置の中で BL 順序の最も小さい位置に, 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 3 次元における BL 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考える．

ある直方体 i を BL 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．以上より, 既配置の直方体の前面と容器の背面の座標の値に z_i の候補を絞って探索すればよいことが分かる．

これらの候補の各値 z'（例えば直方体 j の前面であれば $z' = z_j + d_j$ ）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．現在注目している値 z' における X-Y 平面と重複をもつ既配置の直方体のおのおの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, 未配置の直方体の背面のおのおのに対して, 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．1 つ以上の直方体に対して BL 点が見つかった場合は, BL 順序の意味で最も小さい BL 点に, そこ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．（既配置の直方体全てについて, それらの背面の Z 軸の値は 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障できる.）そして, 次の直方体の配置位置の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．

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 best-fit 法で配置するのに要する計算時間を考える．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については, 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．

3.4 アルゴリズムの実用的高速化

ここでは 3.3 節で提案したアルゴリズムの実用的な高速化について議論する．

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 (x_i, y_i) の候補を, 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 BL 順序の昇順に行っていく．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 BL 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 BL 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 BL 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, 値 z' 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, 直方体の前面の座標が z' である直方体が多数あるときに, その多くに対して Find2D-BL を呼び出す必要が生じる場合もあり, 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．そこで, 2 つのアイデアを以下のように組み合わせることで, 最悪計算量の増加を防ぐ．ある z' の値に対し, ある未配置の直方体の BL 点を探すことを考える．まず最初に, Find2D-BL の探索範囲を直方体の NFP の前面領域に絞る方法を試みる．これを z' に前面をもつ前面領域の各々について上述の順序で調べていくが, その探索途中で（現在の z' と未配置の直方体の組に対する）これまでの総計算時間（具体的には計算対象とした既配置の直方体の延べ数に基づいて計る）が探索範囲を X-Y 平面全体にする場合の最悪計算時間の定数倍を越えたときに, 探索範囲を X-Y 平面全体とする方法に切り替える．これにより, 計算時間を $O(tn^2 \log n)$ に抑えられることが保証でき, また実用的

な高速化が期待できる．

3.3 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, 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ためここでは 1 としておく)． 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．よって, 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 BL 順序の意味で小さいか等しい．BL_rを用いて上述の 2 つの高速化を行う方法を以下に述べる．

まず, BL_rを用いて Find2D-BL の呼び出し回数を減らす方法を説明する．BL_rは未配置の直方体のいずれの BL 点よりも BL 順序の意味で小さいか等しい．よって, 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, そのような直方体の中で, 優先順位が最も高いものをBL_rに配置すればよい．また, r を置けないときには現在の X-Y 平面には未配置の直方体のいずれも配置できない．よって, 現在の面に対して未配置の直方体の BL 点の探索を行う必要がない．このことから次の高速化法を考える．現在の X-Y 平面に対して r の BL 点を探索し, BL 点が見つかった場合はそれをBL_rとする．未配置の直方体を優先順位の高い順に並べ, その順序に従って BL 点を探索する．初めて未配置の直方体の BL 点がBL_rと一致したとき, その直方体をBL_rに配置する．この直方体より優先順位の高い直方体はBL_rに配置できない．よって, この直方体はBL_rに配置可能な直方体の中で最も優先順位の高い直方体となる．一方, r の BL 点が見つからない場合は, 次の X-Y 平面の探索に移る．こうすることで, 多くの場合, 全ての未配置の直方体に対して BL 点の探索を行う必要がなくなる, または現在の面を探索せずに済むために Find2D-BL の呼び出し回数の削減が期待できる．

次にBL_rを用いて Find2D-BL の探索範囲を絞る方法を説明する．BL_rが決定したとき, 現在の X-Y 平面においてBL_rより BL 順序の意味で小さい位置には未配置の直方体のいずれも配置不可能である．よって, 未配置の直方体の BL 点の探索は, X-Y 平面のうちBL_rの Y 座標以上の部分で行えばよい．このことから次の高速化法を考える． r の BL 点を探索し, BL_rが決定した場合は, 探索範囲をBL_rの Y 座標以上の部分に絞る．これにより, Find2D-BL の探索範囲を絞ることでアルゴリズムの高速化が図れる．

なお, 上述の探索範囲を直方体の NFP の前面領域に絞る方法を用いて r の BL 点を探索する場合は, 探索範囲に注意が必要である． 既配置の直方体に対する r の NFP の前面領域のみを探索したのでは, BL_rを正しく計算できない場合があるからである． そこで, $w^*_{\min} = \min_{h_j = h_{\min} \wedge j \in I_{\mathrm{u}}} w_j$, $h^*_{\min} = \min_{w_j = w_{\min} \wedge j \in I_{\mathrm{u}}} h_j$, $d^*_{\min} = 1$, 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．探索範囲を既配置の直方体に対する r^* の NFP として得られた 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 BL 順序の意味で小さいか等しいことを保証できる．

上記の他にもアルゴリズムを高速化する方法を 2 つ組み込んでいる．1 つ目は, BL 順序の意味で小さい BL 点を持つ可能性の高い, 背面の小さい未配置の直方体をI_{u}の中から複数選んで集合 $I_{\mathrm{u}}^{\mathrm{small}}$ を定め, $I_{\mathrm{u}}^{\mathrm{small}}$ 中の全ての直方体の BL 点を探索したのち, それらの中で BL 順序の意味で最も小さかった BL 点の Y 座標を探索範囲の上限とする方法である．具体的には次の 4 つの直方体を考え, これらの集合を $I_{\mathrm{u}}^{\mathrm{small}}$ とする．

- w_i が最も小さいものの中で h_i が最も小さい直方体 $i \in I_{\mathrm{u}}$,
- h_i が最も小さいものの中で w_i が最も小さい直方体 $i \in I_{\mathrm{u}}$,
- $w_i + h_i$ が最も小さいものの中で w_i が最も小さい直方体 $i \in I_{\mathrm{u}}$,
- $w_i h_i$ が最も小さいものの中で w_i が最も小さい直方体 $i \in I_{\mathrm{u}}$.

ただし, 直方体 $j \in I_{\mathrm{u}}^{\mathrm{small}}$ に対して, $w_i \leq w_j$ かつ $h_i \leq h_j$ を満たす直方体 $i \in I_{\mathrm{u}}^{\mathrm{small}} \setminus \{j\}$ があれば, 集合 $I_{\mathrm{u}}^{\mathrm{small}}$ から直方体 j を除く．これは, 直方体 j の BL 点が直方体 i の BL 点より BL 順序の意味で小さくなることはないことが直ちに分かり, j について BL 点を調べる必要はないからである．2 つ目は, ある面に複数の直方体を配置するときに有効な方法である．未配置の直方体の各形状に対して, 現在の面における前回の BL 点を記憶し, 次の探索はその BL 点の高さから始める．これらにより, Find2D-BL の探索範囲を絞ることができ, 高速化が図れる．

以下ではまず, ある X-Y 平面における 1 つの直方体の BL 点を探索する手続きに上述の高速化のアイデアを組み込んだものの詳細を示す．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の集合$\{j \in I \setminus I_{\mathrm{u}} \mid z_j \leq z' < z_j + d_j\}$を, 前面の座標の BL 順序の昇順に整列したリストを N_{cross}とする． また, 直方体の前面の Z 座標の値が z' と等しい既配置の直方体の集合$\{j \in I \setminus I_{\mathrm{u}} \mid z_j + d_j = z'\}$を, 前面の座標の BL 順序の昇順に整列したリストを N_{on}とする．N_{cross} (N_{on}) の l番目の直方体が p であることを $N_{\mathrm{cross}}(l) = p$

($N_{\rm on}(l) = p$) と表す． $N_{\rm on}$内の既配置の直方体の NFP の前面領域の各々を探索範囲とし，リスト $N_{\rm on}$の順序に従って探索を行っていく．1 つの前面領域の探索が終了したら，その前面領域の探索において対象となった直方体の個数を $n_{\rm cum}$ に加算する．そして，α (> 0) をパラメータとして条件 $n_{\rm cum} \geq \alpha n$ を満たしたときに探索範囲を X-Y 平面全体に切り替える．本稿では $\alpha = 1$ とした．なお，実験中にこの条件が満たされることはなかった．入力として，BL 点を探索する未配置の直方体 i, $N_{\rm cross}$, $N_{\rm on}$, 探索すべき Y 座標の下界 $y_{\rm lb}$ (BL'_rまたは現在の面における前回の i の探索により見つけた BL 点の Y 座標), 上界 $y_{\rm ub}$ （他の未配置の直方体の BL 点で BL 順序の最も小さいものの Y 座標）を与える．次のアルゴリズムを Find2D-BL$^*(i, N_{\rm cross}, N_{\rm on}, y_{\rm lb}, y_{\rm ub})$ と呼ぶ．

Algorithm Find2D-BL$^*(i, N_{\rm cross}, N_{\rm on}, y_{\rm lb}, y_{\rm ub})$

Step 1: $l := 1$, $n_{\rm cum} := 0$, $y'_{\rm ub} := y_{\rm ub}$ とする．暫定 BL 点を $(x, y) = (\infty, \infty)$ とする．

Step 2: $l = |N_{\rm on}|+1$ ならば Step 6 へ．$p := N_{\rm on}(l)$ とする．$y_p - h_i > y'_{\rm ub}$ ならば Step 6 へ．$y_p + h_p < y_{\rm lb}$ ならば，$l := l + 1$ としたのち，Step 2 へ戻る．

Step 3: $n_{\rm cum} < \alpha n$ ならば，NFP(p, i) の前面（ただし，i が r の場合は NFP(p, r^*) の前面）のうち Y 座標が $y_{\rm lb}$ 以上 $y'_{\rm ub}$ 以下の領域を探索領域 S とする．一方 $n_{\rm cum} \geq \alpha n$ ならば，z' が定める X-Y 平面のうち，Y 座標が $y_p - h_i$ より大きく，さらに $y_{\rm lb}$ 以上 $y'_{\rm ub}$ 以下の位置全てからなる領域を探索領域 S とする．

Step 4: リスト $N_{\rm cross}$の中で，NFP(j, i) が探索領域 S と重なりを持つ直方体 j の集合を Eとする．探索領域 S と既配置の直方体集合 Eに対して Find2D-BL を呼び出し，i の BL 点を探索する．

Step 5: S 内に i の BL 点を見つけた場合は，新たに見つけたものと暫定 BL 点のどちらが BL 順序の下で小さいかを比較し，小さい方を暫定 BL 点とする．また，暫定 BL 点の Y 座標と $y'_{\rm ub}$ を比較し，小さい方の値を上界 $y'_{\rm ub}$ に代入する．$n_{\rm cum} < \alpha n$ ならば，$n_{\rm cum} := n_{\rm cum} + |E|$, $l := l + 1$ としたのち，Step 2 へ戻る．

Step 6: 暫定 BL 点が (∞, ∞) ならば，i は注目している平面に配置できないと確定し，終了する．そうでなければ，暫定 BL 点を BL 点と確定し，BL 点 (x_i, y_i) を出力して終了する．

次にアルゴリズム全体の流れを示す．アルゴリズムは，探索範囲となる X-Y 平面を定める z' の候補を小さい順に調べていき，各 z' に対して，未配置の直方体の BL 点を直方体の優先順位の高い順に調べて行く．このとき，得られた BL 点がBL'_rと一致した場合は，その直方体を直ちに配置し，それ以降の直方体の BL 点の探索を省略する．また，各直方体の BL 点の探索には Find2D-BL* を用いるが，その際，Y 軸方向の探索範囲を定める $y_{\rm lb}$ と $y_{\rm ub}$ にこれまでの探索から得られる情報を反映することによって高速化を図る．以下では，直方体 i の前面の Z 座標を $z(i)$ と表す．つまり，$z(i) = z_i + d_i$ である．直方体を優先順位の順に並べた順列を σ とし，順列の s 番目の直方体が i であることを $\sigma(s) = i$ と表す．また，直方体 $i \in I$ の形状が $k \in K$ であることを $\pi(i) = k$ と記す．注目している z' が定める面に対して形状 k の BL 点を一度も探索していなければ $\tau(k) = 0$ とし，探索済みならば $\tau(k) = 1$ とする．また，直方体の各形状 k の下界を $y'_{\rm lb}(k)$ と表す．

Algorithm Fast Best-Fit

Step 1: 集合 $I_{\rm u} := I$ とし，$(0, 0, 0)$ に配置された容器と同じ幅，高さを持ち，奥行きが 0 の直方体のみを含むリスト $N_{\rm cross}$と空のリスト $N_{\rm on}$を用意する．暫定 BL 点を $(x, y, z) = (\infty, \infty, \infty)$ とする．全ての k に対して $\tau(k) := 0$ とする．

Step 2: $N_{\rm cross}$の最初の直方体を p とする．つまり，$p := N_{\rm cross}(1)$ とする．$z' := z(p)$ とする．探索領域の下界を $y_{\rm lb} := 0$, 上界を $y_{\rm ub} := H$ とする．また，直方体の各形状 k の下界を $y'_{\rm lb}(k) := 0$ とする．

Step 3: $z' = z(p)$ ならば，$N_{\rm cross} := N_{\rm cross} \setminus \{p\}$ とし，p を $N_{\rm on}$の末尾に追加したのち，$p := N_{\rm cross}(1)$ として Step 3 へ．そうでないなら，Step 4 へ．

Step 4: Find2D-BL$^*(r, N_{\rm cross}, N_{\rm on}, y_{\rm lb}, y_{\rm ub})$ を実行する．BL 点を見つけたならば，それをBL'_rとし，BL'_r の Y 座標の値を下界 $y_{\rm lb}$ に代入する．一方，r を配置できないならば，Step 9 へ．

Step 5: 各直方体 $i \in I_{\rm u}^{\rm small}$ に対して，$y'_{\rm lb}(\pi(i))$ と $y_{\rm lb}$ を比較し，大きい方を下界 LB として Find2D-BL$^*(i, N_{\rm cross}, N_{\rm on}, LB, y_{\rm ub})$ を実行する．発見した BL 点の中で BL 順序の最も小さいものの Y 座標を上界 $y_{\rm ub}$ とする．$s := 1$ とする．

Step 6: $s = n + 1$ ならば Step 9 へ．順列 σ の s 番目の直方体を $i := \sigma(s)$ としたのちその形状を $k := \pi(i)$ とする．$i \notin I_{\rm u}$ または $\tau(k) = 1$ ならば，$s := s + 1$ とし，Step 6 に戻る．さもなければ Step 7 へ．

Step 7: $y'_{\rm lb}(k)$ と $y_{\rm lb}$ を比較し，大きい方を下界 LB として Find2D-BL$^*(i, N_{\rm cross}, N_{\rm on}, LB, y_{\rm ub})$ を実行し，$\tau(k) := 1$ とする．BL 点を見つけたならば，Step 8 へ．見つからないならば，上界 $y_{\rm ub}$ を

形状 k の下界 $y'_{\mathrm{lb}}(k)$ に代入する．$s := s+1$ とし，Step 6 に戻る．

Step 8: 発見した BL 点の Y 座標の値を形状 k の下界 $y'_{\mathrm{lb}}(k)$ に代入する．発見した BL 点と BL_r が等しいならば，その BL 点を暫定 BL 点とし，暫定直方体を $i^* := i$ として，Step 9 へ．等しくないならば，新たに見つけたものと暫定 BL 点のどちらが BL 順序の下で小さいかを比較し，小さい方を暫定 BL 点とする．新たに見つけた BL 点が暫定 BL 点となる場合は暫定直方体を $i^* := i$ と更新し，暫定 BL 点の Y 座標の値を上界 y_{ub} に代入する．$s := s+1$ とし，Step 6 に戻る．

Step 9: 各 k に対して $\tau(k) := 0$ とする．暫定 BL 点が (∞, ∞, ∞) ならば，N_{on} を空にし，Step 2 へ．そうでなければ，暫定 BL 点を BL 点と確定し，直方体 i^* を BL 点に配置する．N_{cross} の適切な位置に直方体 i^* を追加し，$I_{\mathrm{u}} := I_{\mathrm{u}} \setminus \{i^*\}$ とする．$I_{\mathrm{u}} = \emptyset$ ならば Step 10 へ．そうでないなら，I_{u}の更新に伴い直方体 r と r^*，および直方体集合 $I_{\mathrm{u}}^{\mathrm{small}}$ を更新する．下界 y_{lb} に BL 点の Y 座標の値を代入し，上界を $y_{\mathrm{ub}} := H$ とする．暫定 BL 点を $(x, y, z) = (\infty, \infty, \infty)$ とし，Step 4 へ．

Step 10: 各直方体 $i \in I$ の座標 (x_i, y_i, z_i) および目的関数値 $D = \max_{i \in I}(z_i + d_i)$ を出力して終了する．

4 計算実験

計算実験により，文献 [14] で提案されている 3 次元の bottom-left 法との解の質と計算時間の比較を行った．また，高速化を行わないアルゴリズム（すなわち 3.3 節のアルゴリズム）との計算時間の比較によって，3.4 節の実用的高速化の有効性を確認した．計算実験は全て Dell Precision 470 (Xeon 3GHz, 1MB cache, 1GB memory) 上で行った．

実験には，文献 [14] で用いられている問題例を使用した．この問題例は，元となる大きな 1 つの直方体から始めて，ギロチンカット（直方体を 1 つの平面で 2 つの直方体に分割するようなカット）による分割を繰り返すことによって複数の直方体に分割する方法を用いてランダムに生成されている．元の直方体と同じ形の背面を持つ容器に対して，分割した直方体の詰め込みを考えると，最適値は元の直方体の奥行きと一致する．

分割する際には，ρ と γ をパラメータとして，生成されるどの直方体も，3 辺のうち長さが最大のものと最小のものの比が ρ 以下であり，しかも生成される直方体のどの 2 つもそれらの体積比が γ 以下になるように問題例が生成されている．これらの問題例では，全ての直方体が相違なる形をしているため，$t = n$ となる．よって，実験に用いた問題例に対する提案手法の最悪計算量は $O(n^3 \log n)$ 時間となる．

表 1: 提案手法と bottom-left 法 [14] との比較

	best-fit 法		bottom-left 法	
n	VU (%)	time (sec)	VU (%)	time (sec)
1000	78.2	2.2	81.6	1.5
2000	83.9	8.7	83.5	4.8
3000	86.9	19.2	85.0	10.5
4000	88.8	31.2	85.5	20.0
5000	90.4	63.6	86.3	30.1
6000	89.9	74.2	86.3	39.6
7000	91.2	118.1	87.2	53.8
8000	91.6	144.2	86.3	67.1
9000	92.2	206.6	87.0	93.8
10000	92.6	221.9	87.0	108.0

まず，3 次元の bottom-left 法との比較を表 1 に示す．表 1 に結果を示した実験では，パラメータを $\rho = 3$, $\gamma = 10$ とし，直方体の数 n が $1000, 2000, \ldots, 10000$ の 10 通りの場合それぞれに対して 5 問ずつ，計 50 問を使用した．表中，VU (%) と time (sec) は，それぞれ充填率と計算時間（秒）の平均値である．VU (%) は次の式で与えられる:

$$\mathrm{VU} = \frac{\sum_{j \in I} w_j h_j d_j}{WHD} \times 100.$$

3 次元の bottom-left 法は，直方体の順列が与えられ，その順列に従って各直方体をその BL 点に配置することを繰り返す方法である．この手法は，詰め込む順序が解の精度に影響を与える．本実験では，詰め込み順序は文献 [14] 中で最も良い結果を示している直方体の奥行き d_i の降順を用いた．文献 [14] で提案されたアルゴリズムの最悪計算量は $O(n^3 \log n)$ 時間である．

表 1 より，bottom-left 法に比べて提案手法は約 2 倍の計算時間がかかることが分かる．しかし，直方体数 1000 を除く問題例に対して提案手法は 3 次元の bottom-left 法より解の精度は良い．提案手法は直方体数が多いほど充填率が高いことが観測できる．

次に，3.4 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，高速化なし（すなわち 3.3 節のアルゴリズム）と高速化あり（すなわち 3.3 節のアルゴリズムに 3.4 節の変更を加えたもの）の比較を表 2 に示す．実験には表 1 に結果を示した実験に利用した問題例と同じ 50 問を使用した．表中，VU (%) と time (sec) は，それぞれ充填率と計算時間（秒）の平均値である．表より，アルゴリズムの高速化の効果は非常に大きく，9 割以上の計算時間を削減することができたことを確認できる．

5 まとめ

2 次元の長方形詰め込み問題に対する代表的な構築型解法である best-fit 法は，3 次元箱詰め問題に拡張できる．本研究では，この解法を $O(tn^2 \log n)$ の計算時間で実現するアルゴリズムを提案した．

表 2: アルゴリズムの実用的高速化の効果

		time (sec)	
n	VU (%)	高速化無し	高速化有り
1000	78.2	29.5	2.2
2000	83.9	137.1	8.7
3000	86.9	314.3	19.2
4000	88.8	557.6	31.2
5000	90.4	947.4	63.6
6000	89.9	1308.8	74.2
7000	91.2	1848.2	118.1
8000	91.6	2299.3	144.2
9000	92.2	3139.7	206.6
10000	92.6	3684.0	221.9

直方体数最大 10000 の問題例に対して計算実験を行った結果, 大規模な問題例に対しても数百秒程度と実用的な時間で解を構築できることを確認できた. また, 提案した $O(tn^2 \log n)$ 時間のアルゴリズムに, 不要な探索を省略するアイデアを導入することによってさらなる効率化を図り, 計算時間を 9 割以上削減できることを計算実験によって確かめた.

本研究では直方体数最大 10000 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. best-fit 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 bottom-left 法等を用いて改善していくことにより, 解の精度の向上を目指していきたい.

参考文献

[1] H. Dyckhoff: A typology of cutting and packing problems, European Journal of Operational Research, 44 (1990), 145–159

[2] G. Wäscher, H. Haußner, H. Schumann: An improved typology of cutting and packing problems, European Journal of Operational Research, 183 (2007), 1109–1130

[3] 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

[4] E. K. Burke, G. Kendall, G. Whitwell: A new placement heuristic for the orthogonal stock-cutting problem, Operational Research, 52 (2004), 655–671

[5] S. Imahori, M. Yagiura: The best-fit heuristic for the rectangular strip packing problem: an efficient implementation and the worst-case approximation ratio, Computer & Operations Research, 37 (2010), 325–333

[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] A. Bortfeldt, H. Gehring: A hybrid genetic algorithm for the container loading problem, European Journal of Operational Research, 131 (2001), 143–161

[8] A. Lodi, S. Martello, D. Vigo: Heuristic algorithms for the three-dimensional bin packing problem, European Journal of Operational Research, 141 (2002), 410–420

[9] J. Egeblad, D. Pisinger: Heuristic approaches for the two- and three-dimensional knapsack packing problem, Computers & Operations Research, 36 (2009), 1026–1049

[10] A. Bortfeldt, D. Mack: A heuristic for the three-dimensional strip packing problem, European Journal of Operational Research, 183 (2007), 1267–1279

[11] 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

[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), 九州大学伊都キャンパス, 2010 年 9 月 7–9 日, 講演論文集 (第 1 分冊), pp.25–30

[13] A. M. Gomes, J. F. Oliveira: A 2-exchange heuristic for nesting problems, European Journal of Operational Research, 141 (2002), 359–370

[14] 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), 九州大学伊都キャンパス, 2010 年 9 月 7–9 日, 講演論文集 (第 1 分冊), pp.31–38