RA-006

3 次元箱詰め問題に対する構築型解法の効率的実現法
An efficient implementation of a constructive algorithm for the three-dimensional packing problem

川島大貴* Hiroki Kawashima　田中勇真* Yuma Tanaka　今堀慎治† Shinji Imahori　柳浦睦憲* 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 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 [2]. 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 bottom-left (BL) 法 [3] がある. BL 法とは, 初めに長方形を詰め込む順番を定め, その順に従って各長方形をなるべく下, 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 に対して $O(n^4)$ 時間を要してしまうが, データ構造を工夫することで $O(n^2 \log n)$ 時間 [4, 5] あるいは $O(n^2)$ 時間 [6] で実行できることが知られている. 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 [3]. 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ことが知られており，また, メタ戦略を用いて良い順番を探索することで, より精度の高い解を見つける試みも行われている [7]. 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点（BL 安定点）の 1 つに配置する BL 法に近いアルゴリズムも存在する [8, 9].

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 [10] 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 [11] 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 Pissinger [12] 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 [13] は 3 次元ストリップパッキング問題に対して, コンテナ詰め込み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 次元箱詰め問題を解くのに用いられる BL 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番に従い, 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 3 次元の BL 法と呼ぶ (3 次元であることが明らかな場合は単に BL 法と呼

*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

ぶ). なお, 2 次元ストリップパッキング問題において隙間のない最適解（完全パッキング (perfect packing) と呼ばれる）が存在する問題例に対しては, BL 法によって最適解が生成される長方形の順列が少なくとも 1 つ存在する [14]. 同様の性質が 3 次元の BL 法に対しても成り立つことを容易に示すことができる.

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るが, 本稿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.

本稿では, 3 次元における BL 法を直方体数 n に対して $O(n^3 \log n)$ 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.

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. 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または容器の幅, 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 の最も奥かつ底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　$D \to$ 最小化

制約条件

$$0 \le x_i \le W - w_i, \quad \forall i \in I, \tag{1}$$

$$0 \le y_i \le H - h_i, \quad \forall i \in I, \tag{2}$$

$$0 \le z_i \le D - d_i, \quad \forall i \in I, \tag{3}$$

任意の相違なる直方体対 $i, j \in I$ が次の 6 つの不等式の少なくとも 1 つを満たす: (4)

$$x_i + w_i \le x_j, \quad x_j + w_j \le x_i,$$
$$y_i + h_i \le y_j, \quad y_j + h_j \le y_i,$$
$$z_i + d_i \le z_j, \quad z_j + d_j \le z_i.$$

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, Imahori らの論文 [5] 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 の例である.

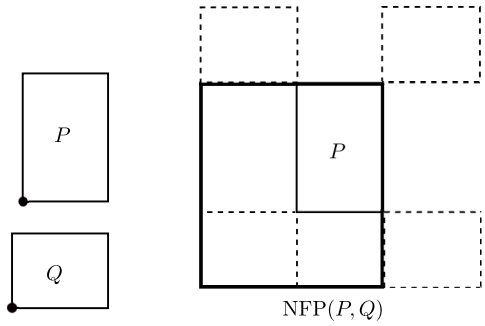

図 1: NFP の例

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,

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

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 (Find2D-BL)

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．Imahori らの提案する 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, Find2D-BL 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．（これは本研究で提案するアルゴリズムが動作するために必要な性質であるが，文献 [4] や [6] のアルゴリズムは既配置の長方形同士が重複を持たないことを前提として設計されているため，利用できない.）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 [15].）これを用いると, 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.

Imahori らのアルゴリズムは, 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保持しておく．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, 及び BL 点である.

既配置の長方形の数を k 個とすると, Find2D-BL は $O(k \log k)$ 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 法

本節では, 3 次元における BL 点の定義を説明した後, それを効率的に発見する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

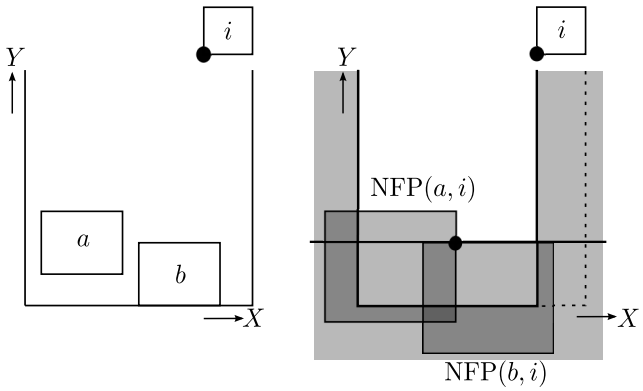

図 2: 2 次元の BL 点の例

$P_i = (x_i, y_i, z_i)$ 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 を既配置のもの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.

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しているはずである．つまり，i は既配置の直方体の前面または容器の背面に i の背面が接する位置の中で配置可能なもののうち, Z 座標が最も小さい面に配置される．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 BL 点を探索すれば良いことが分かる．z_i の候補となる各値 z' （例えば直方体 j の前面であれば $z' = z_j + d_j$ ）に対しては, 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 平面に射影したのち，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 点を探せばよい.

既配置の直方体の前面と容器の背面を面の座標の BL 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 座標が同じものが探索の対象となる面の中にないときは, 見つけた BL 点が i の 3 次元における BL 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 3 次元における BL 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, BL 順序の意味で最も小さい座標が 3 次元の BL 点となる.

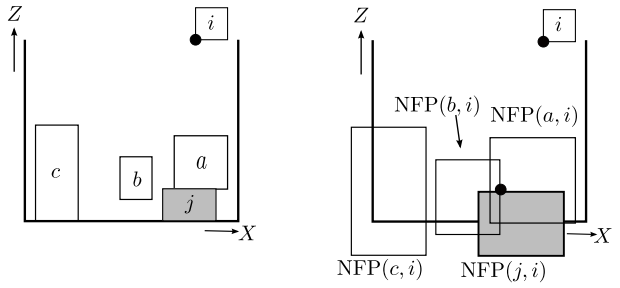

図 3: 3 次元の NFP

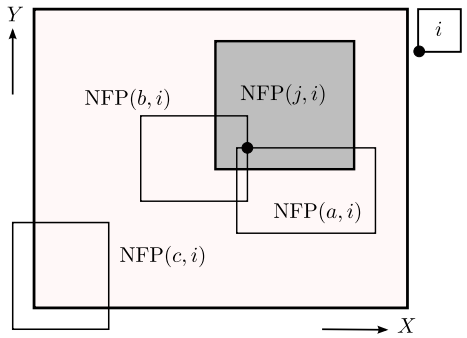

図 4: NFP(j, i) の前面における断面図

BL 点を発見するために, 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)．このような候補の中で，i が他のどの直方体とも重複を持たないための条件はどの NFP の内部にも含まれないことであるが, これは平面 $z = z_j + d_j$ で切った 2 次元平面で考えれば良い．図 3 の左右の図はそれぞれいくつかの直方体（左）及びそれらの NFP（右）を X-Z 平面に射影した例である．図 4 は, 図 3 の NFP(j, i) の前面で切った時の Z 軸の上方向から見た断面図の例であり, NFP(j, i) の前面領域内における BL 点は, 灰色領域内の黒点である．

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, Imahori らのアルゴリズム Find2D-BL を使用することにより, $O(k \log k)$ の計算時間で 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 BL 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 3 次元の BL 点が求まる．

直方体 j に対する i の NFP の前面領域と NFP が重なる直方体を全て見つける操作を，j が変わる度にその都度行うと, 既配置の直方体 m 個のおのおのに対して, すなわち Find2D-BL を呼び出す度にその前処理に $O(m)$ の時間がかかってしまう．k は最大で m になり得るが, 多くの場合 $k \ll m$ であることが予想される．そこで, この問題を回避するために以下に定義する集合 E を使用する．直方体 i を配置するとき, 既配置の直方体を, 前面の座標 $z_j + d_j$ の BL 順序の昇順に整列したリストと背面の座標 z_j の BL 順序の昇順に整列したリストをそれぞれ N_f と N_b とする．N_f の l_f 番目の直方体が j_f であることを $N_f(l_f) = j_f$，N_b の l_b 番目の直方体が j_b であることを $N_b(l_b) = j_b$ と表す．$E := \emptyset$, $l_f := l_b := 1$, $j_f := N_f(l_f)$, $j_b := N_b(l_b)$ から始め, 各反復では, NFP の前面の座標 $z_{j_f} + d_{j_f}$ と背面の座標 $z_{j_b} - d_i$ の大小を比較する．背面の値が小さい場合は j_b を E に追加し, $l_b := l_b + 1$ とした後 $j_b := N_b(l_b)$ とする．一方, 両者が等しい, あるいは前面の値が小さい場合は j_f を E から除き, $l_f := l_f + 1$ とした後 $j_f := N_f(l_f)$ とする．なお，l_b が $m + 1$ に達した後は, 前面 j_f を E から除く操作のみを行う．E から j_f が除かれた直後，E には Z 座標が j_f の前面と同じ値の X-Y 平面に NFP が重なっている直方体が入っている．この中から j_f の前面と重なりを持つ直方体を取り出した後, それらに対して Find2D-BL を呼び出す．通常 $|E| \ll m$ であることが多いため, 既配置の直方体全てから探すより計算時間が短くなる．集合 E の管理に伴う計算時間は, Find2D-BL を m 回呼び出す全反復を通して $O(m)$ である．図 4 において j が E から除かれた時点では, 直方体 a から c が E に入っている．NFP(j, i) の前面と重なりを持つ直方体 a と b を E から取り出し, Find2D-BL を呼ぶ．

3 次元の BL 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．本研究では, 適当な指標による順列が与えられ, その順に詰め込みを行うものとする (順列を定める指標については 4 章を参照)．

以下に BL 法のアルゴリズム全体の流れを示す．直方体集合 $I = \{1, 2, \ldots, n\}$ に対して与えられる順列を 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．容器の背面に直方体 i の BL 点があるか否かをまだ確認していないときは $floor = 0$ とし, 確認済みであるときは $floor = 1$ とする．また, 既配置の直方体の前面に対して, その座標を前面の中で X, Y 座標が最も小さい頂点の座標（つまり左下の頂点の座標）とする．

Step 1: 集合 $E := \emptyset$, および $(0, 0, \infty)$ の座標に配置されている面のみを含むリスト N_f と N_b を用意する．$s := 1$ とする．

Step 2: $s = n + 1$ ならば Step 10 へ．そうでなければ, 順列 σ の s 番目の直方体 $i = \sigma(s) \in I$ を取

り出したのち, $s := s + 1$ とする．i の暫定 BL 点を $(x_i, y_i, z_i) = (\infty, \infty, \infty)$ とし, $l_f := l_b := 1$, $j_f := N_f(l_f)$, $j_b := N_b(l_b)$, $floor := 0$ とする．

Step 3: 既配置の直方体 j のおのおのに対して，i の NFP すなわち NFP(j, i) を作成する．

Step 4: $floor = 0$ の時, NFP(j_b, i) の背面の Z 座標 $z_{j_b} - d_i$ と容器の背面の Z 座標 $(z = 0)$ の大小を比較し, $z_{j_b} - d_i < 0$ ならば Step 5 へ, $z_{j_b} - d_i \geq 0$ ならば Step 8 へ進む. 一方, $floor = 1$ の時は, NFP(j_b, i) の背面の Z 座標 $z_{j_b} - d_i$ と NFP(j_f, i) の前面の Z 座標 $z_{j_f} + d_{j_f}$ の大小を比較し, $z_{j_b} - d_i < z_{j_f} + d_{j_f}$ ならば Step 5 へ, $z_{j_b} - d_i \geq z_{j_f} + d_{j_f}$ ならば Step 6 へ進む.

Step 5: $E := E \cup \{j_b\}$, $l_b := l_b + 1$ とした後, $j_b := N_b(l_b)$ とし, Step 4 に戻る.

Step 6: $E := E \setminus \{j_f\}$ とする． 集合 E の中で, NFP(j_f, i) の前面と NFP が重なりを持つ直方体の集合を E' とし, E' に対して NFP(j_f, i) の前面領域で Find2D-BL を呼び出す．NFP(j_f, i) の前面領域内に 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.

Step 7: リスト N_f の次の直方体 $N_f(l_f + 1)$ に対する i の NFP の前面の座標が, 暫定 BL 点より BL 順序において小さい場合は, $l_f := l_f + 1$ とした後 $j_f := N_f(l_f)$ とし, Step 4 に戻る．一方, 暫定 BL 点の方が小さいか等しい場合は, 暫定 BL 点を 3 次元における BL 点と確定し, Step 9 に進む.

Step 8: 集合 E に対して容器の背面で Find2D-BL を呼び出し, $floor := 1$ とする．容器の背面上に BL 点を発見できなかった場合は Step 4 に戻る．一方, BL 点を発見した場合は, その点を 3 次元における BL 点と確定し, Step 9 に進む.

Step 9: 3 次元における BL 点に i を配置し，リスト N_f （ N_b ）の適切な位置に i の前面（背面）を追加する (つまり, 挿入後のリストが BL 順序に従うような位置に追加する). Step 2 に戻る.

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.

m 個の既配置の直方体があるときに, 新たな直方体の配置にかかる計算時間は $O(m^2 \log m)$ である．よって, n 個の直方体を詰め込むことは $O(n^3 \log n)$ の計算時間で可能である.

3.4 アルゴリズムの高速化

容器に直方体を配置し続けていくと, いくつかの既配置の直方体の前面と容器の背面の中には, 未配置の直方体のいずれも配置できない面が存在する可能性がある．このような面を探し出し, 探索する候補から除外することにより計算時間の向上を考える．現在の未配置の直方体の集合を $I_u \subseteq I$ と記し, $w_{\min} = \min_{j \in I_u} w_j$, $h_{\min} = \min_{j \in I_u} h_j$, $d_{\min} = \min_{j \in I_u} d_j$ 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． r は未配置の直方体いずれよりも小さいか等しい．よって，r を置くことができない直方体の前面と容器の背面には未配置の直方体のいずれも置くことができないことが結論できる．この条件を満たすことが判明した面の集合を N_e と記す.

既配置の直方体の前面と容器の背面に対して Find2D-BL を使用し，r が配置できないと確認された面を N_e に追加する．全ての直方体の前面と容器の背面に対してこれを確認する必要はなく, 前回の確認後に配置された直方体により，r が置けなくなった可能性のある直方体の前面と容器の背面のみ確認すれば良い．もちろん, 既に候補から除外されている直方体の前面と容器の背面は確認する必要はない．また，r の大きさが更新されたときは, 候補から除外されていない全ての直方体の前面と容器の背面に対して確認を行う.

探索の候補となる面のうち, 新たに N_e に入る面があるかどうかを確認する作業には, 1 つの直方体を配置するのに要する時間と同等の計算時間が必要であるため, 毎反復ごとにこの作業を行うのは逆効果である．そこで, 確認作業に費やす時間が, BL 点の探索に費やす時間の高々定数倍程度で収まるような頻度で確認作業を行うことで探索の効率化を図る.

具体的には, 次の条件を満たすときに確認を行うことを考えた．Find2D-BL の計算時間は, 3.3 節のアルゴリズムの計算時間の大部分を占める．よって, Find2D-BL を行う回数により, 計算時間を大まかに見積もることができると考えられる． I_pを既配置の直方体の集合 (つまり $I_p = I \setminus I_u$)とすると, $(|I_p| + 1 - |N_e|)$ は確認作業において Find2D-BL を呼び出す対象となる直方体の前面と容器の背面の数の上界を表す． この値を, 前回確認作業を行った時点以降に BL 点探索のために Find2D-BL を呼び出した回数（これを$NumFind$ と記す）と比較し，$\alpha(> 0)$ をパラメータとして条件 $\alpha(|I_p| + 1 - |N_e|) < NumFind$ が満たされるときに限って確認作業を行う．こうすることで，本節で提案する高速化の効果が得られない（つまり N_e が空集合に近い）ような問題例に対しても，確認作業を行わない場合の $1 + 1/\alpha$ 倍程度の計算時間で収まることが期待できる．なお，r の大きさが更新されたときは，r を配置できない直方体の前面と容器の背面が大幅に増える可能性があるため, 上述のルールよりもやや早い段階で確認作業を行えるよう, αの代わりにパラメータ

$\beta = [0, \alpha]$ を使用する．以上をやや詳しくまとめておく．具体的には，3.3 節のアルゴリズムの Step 1, 6, 8, 9 を以下のように修正することにより高速化を実現する．

Step 1: 集合 $E := \emptyset$, および $(0, 0, \infty)$ の座標に配置されている面のみを含むリスト N_f と N_b を用意する．$s := 1$, $I_p := \emptyset$, $I_u := I$, $NumFind := 0$ とする．

Step 6: $E := E \setminus \{j_f\}$ とする．j_f の前面が N_e に含まれないならば, 集合 E から直方体 j_f の NFP の前面と NFP が重なりを持つ直方体を取り出し, それらに対して j_f の NFP の前面領域で Find2D-BL を呼び出したのち, $NumFind := NumFind + 1$ とする．j_f の NFP の前面領域内に BL 点を見つけた場合は, 暫定 BL 点とどちらが BL 順序の意味で小さいかを比較し, 小さい方を暫定 BL 点とする．一方，j_f の前面が N_e に含まれるならば, BL 点の探索は行わない．

Step 8: 容器の底面が N_e に含まれないならば, 集合 E に対して容器の背面で Find2D-BL を呼び出し, $floor := 1$, $NumFind := NumFind + 1$ とする．容器の背面上に BL 点を発見できなかった場合は Step 4 に戻る．一方, BL 点を発見した場合は, その点を 3 次元における BL 点と確定し, Step 9 に進む．一方, 容器の底面が N_e に含まれるならば, $floor := 1$ とし, Step 4 に戻る．

Step 9: 3 次元における BL 点に i を配置し，リスト N_f（N_b）の適切な位置に i の前面（背面）を追加する (つまり, 挿入後のリストが BL 順序に従うような位置に追加する)．$I_u := I_u \setminus \{i\}$, $I_p := I_p \cup \{i\}$ としたのち, I_uの更新によって $w_{\min} = \min_{j \in I_u} w_j$, $h_{\min} = \min_{j \in I_u} h_j$, $d_{\min} = \min_{j \in I_u} d_j$ を幅, 高さ, 奥行きとして持つ直方体 r が変化したか否かを確認する．r に変化がない場合は $\alpha(|I_p|+1-|N_e|) < NumFind$ を満たすとき, 一方 r が更新された場合は $\beta(|I_p|+1-|N_e|) < NumFind$ を満たすとき, 探索の候補となる面（I_pに含まれる直方体の前面と容器の背面から N_e に含まれる面を除いたもの）のおのおのに対して直方体 r を配置可能か否かの確認を行い，N_e を更新したのち, $NumFind := 0$ とする．Step 2 に戻る．

4 計算実験

計算実験により, 詰め込み順序による解の質と計算時間の比較を行った．また, Find2D-BL を用いない単純な 3 次元の BL 法との計算時間の比較を行い, アルゴリズムの性能を調べることによって, 高速化提案手法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．

実験に使用する問題例は, 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した．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 以下になるようにした．

まず, 詰め込み順序の比較を表 1 に示す．この表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 問ずつ生成し, 計 50 問を使用した．BL 法に用いる直方体の順序として, ランダムな順列 Random, 奥行き d_i の降順 D-sort, 体積 $w_i h_i d_i$ の降順 V-sort, 背面の面積 $w_i h_i$ の降順 S-sort の 4 通りを調べた．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．

詰め込み順序が計算時間にも大きく影響を与えることが表から観測できる．これらの問題例に対しては, D-sort による解の精度が最も良いことが分かった．D-sort は他と比べて全体的に計算時間が速く, $n = 10000$ の問題例に関しては他の約 4 倍速いことが分かる．また, 大規模な問題例に対しても, 計算時間は数百秒程度と実用的な時間で詰め込みが行えている．

次に単純な 3 次元の BL 法との計算時間の比較を表 2 に示す．i を BL 点に配置した時，i の X 座標の左側に接するように既配置の直方体あるいは容器が存在するので, 既配置の直方体 m 個に対して BL 点の X 座標の候補は高々$(m+1)$ 個ある．Y 座標および Z 座標についても同様である．したがって, BL 点の候補は $O(m^3)$ 個存在する．単純な BL 法とは, BL 点の候補を BL 順序の最も小さいものから順に，i 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．そのおのおのに対して重なりの確認は $O(m)$ 時間で可能である．長方形を 1 つ配置するたびに上述の計算を行うことにより, n 個の直方体の詰め込みは $O(n^5)$ の計算時間で可能である．このアルゴリズムを Simple-Comp, 本稿で提案したアルゴリズムを NFP-Comp と呼ぶ．（Simple-Comp では, $O(m^3)$ の BL 点の候補を BL 順に調べていくため, 配置可能な場所が見つかった時点で残りの候補の探索を省略できる．その結果, 実際には最悪の場合の計算時間の見積もりよりも高速である可能性がある．一方, この順序を任意とし, 暫定 BL 点を更新するというより単純で実装の容易な方法をとると, このような効果は期待できず, $\Omega(n^5)$ 時間を要してしまう.）

表 1: BL 法に与える詰め込み順序による比較

	Random		D-sort		V-sort		S-sort	
n	VU(%)	time(sec)	VU(%)	time(sec)	VU(%)	time(sec)	VU(%)	time(sec)
1000	70.1	3.5	81.6	1.5	75.7	3.6	69.7	2.7
2000	75.2	14.9	83.5	4.8	77.2	15.8	74.2	9.5
3000	75.4	36.0	85.0	10.5	78.1	37.2	76.8	29.8
4000	76.8	64.3	85.5	20.0	78.3	68.2	79.4	66.5
5000	77.5	104.7	86.3	30.1	78.3	117.1	78.5	96.2
6000	77.4	153.7	86.3	39.6	78.2	165.0	75.3	125.5
7000	78.0	216.0	87.2	53.8	78.7	226.4	77.3	151.0
8000	77.7	256.7	86.3	67.1	77.7	301.0	75.9	189.6
9000	78.3	360.6	87.0	93.8	78.6	390.8	77.9	318.5
10000	78.1	448.8	87.0	108.0	78.2	485.9	78.8	381.5

表 2: Simple-Comp と NFP-Comp の計算時間

		time(sec)	
n	VU(%)	Simple-Comp	NFP-Comp
50	70.19	0.04	0.00
100	73.88	1.83	0.01
150	74.76	14.34	0.03
200	74.88	70.93	0.04
250	75.92	231.34	0.06
300	76.43	621.23	0.08
350	77.10	1485.67	0.11
400	77.40	2912.84	0.16
450	77.36	5974.18	0.19
500	77.19	10493.32	0.23

表 3: 高速化無しとの計算時間の比較

		time(sec)	
n	VU(%)	高速化無し	高速化有り
1000	81.6	8.17	1.49
2000	83.5	37.96	4.84
3000	85.0	90.19	10.45
4000	85.5	162.98	20.01
5000	86.3	262.98	30.14
6000	86.3	382.75	39.55
7000	87.2	523.51	53.76
8000	86.3	690.20	67.14
9000	87.0	837.00	93.84
10000	87.0	1090.82	108.03

表 2 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 $50, 100, \ldots, 500$ の 10 通りの場合それぞれに対して 5 問ずつ生成した. また, BL 法に用いる詰め込み順序には D-sort を用いた．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．なお, Simple-Comp と NFP-Comp の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 NFP-Comp と Simple-Comp の計算時間の差は大きくなっていく．$n = 500$ の時, NFP-Comp の計算時間は 0.2 秒程度であるのに対して, Simple-Comp の計算時間は約 3 時間かかっている.

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, 高速化無し（すなわち 3.3 節のアルゴリズム）と高速化有り（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 3 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．また, BL 法に用いる詰め込み順序には D-sort を用いた．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 割近く計算時間を短縮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 BL 法は，3 次元箱詰め問題に自然に拡張できる．本研究では，この解法を $O(n^3 \log n)$ 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 $O(n^5)$ 時間のものしか存在せず, 大きな改善といえる.

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, $O(n^5)$ 時間で動作するアルゴリズム Simple-Comp に比べて格段に速い計算時間で解を構築でき, 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 $O(n^3 \log n)$ 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 割近く削減できることを計算実験によって確かめた．詰め込み順序に関しては, 簡潔な

規則に基づく 4 通りのルールを比較し, 解の精度, 計算時間共に, 奥行き d_i の降順に詰め込む D-sort が良いことが分かった.

本研究では 4 種類の順列に基づく BL 法によって得られる解の精度を調べたが, 得られた配置の充填率は 8 割程度であり, まだ改善の余地がある. 今後は, NFP-Comp を使用した局所探索等を考え, 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, Europian Jornal of Operational Research, 44 (1990), 145–159

[2] 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

[3] B.S. Baker, E.G. Coffman Jr., R.L. Rivest: Orthogonal packing in two dimensions, SIAM Journal on Computing, 9 (1980), 846–855

[4] P. Healy, M. Creavin, A. Kuusik: An optimal algorithm for rectangle placement, Operations Research Letters, 24 (1999), 73–80

[5] S. Imahori, Y. Chien, Y. Tanaka, M. Yagiura: Enumerating bottom-left stable positions for rectangles with overlap, submitted for publication, 2010

[6] B. Chazelle: The botom-left bin-packing heuristic: an efficient implemention, IEEE Transactions on Computers, C-32 (1983), 697–707

[7] E. Hopper, B.C.H. Turton: A review of the application of meta-heuristic algorithms to 2D strip packing problems, Artificial Inteligence Review, 16 (2001), 257–300

[8] S. Jakobs: On genetic algorithms for the packing of polygons, European Journal of Operational Research, 88 (1996), 165–181

[9] 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

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

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

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

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

[14] N. Lesh, J. Marks, A. McMahon, M. Mitzenmacher: Exhaustive approaches to 2D rectangular perfect packings, Information Processing Letters, 90 (2004), 7–14

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