Vol.2019-AL-172 No.6
2019/3/5

三次元レクトリニア多面体配置問題に対する構築型解法の配置戦略および効率的実現法

梅田 知樹[1,a)]　胡 艷楠[1,b)]　柳浦 睦憲[1,c)]

概要：三次元の多面体で各面が xy 平面, yz 平面, zx 平面のいずれかと平行であるものを三次元レクトリニア多面体と呼び, そのような多面体を直方体の容器に詰め込む問題を三次元レクトリニア多面体配置問題という. 本研究では, 二次元レクトリニア図形配置問題に対する成果を三次元空間に拡張し, 三次元レクトリニア多面体配置問題に対する構築型解法を提案する. また, 高度なデータ構造を組み込む新たなアルゴリズムを開発し, その理論的な計算量を評価する. さらに, より性能を上げるために新たな配置戦略も提案する. 計算実験により, 提案したアルゴリズムの性能を検証した. その結果, 問題例によっては計算時間が 1/8 程度に短縮されることを確認した. また, 計算結果に基づき, 提案する構築型解法において高い充填率を実現するルールの組み合わせを提案する.

1. はじめに

配置問題とは, いくつかの対象物と領域が与えられたとき, その領域内に対象物を互いに重なりなく配置する問題である. この問題は対象物および領域の形状, その次元, 目的関数などによって様々な種類に分類することができる. 本研究では, 三次元レクトリニア多面体配置問題について考える. 三次元レクトリニア多面体配置問題とは, 複数の三次元レクトリニア多面体と 1 つの直方体の容器が与えられたとき, 全ての三次元レクトリニア多面体を容器の中に重ならないように配置する問題である. 三次元レクトリニア多面体とは, 三次元の多面体であり, 各面が xy 平面, yz 平面, zx 平面のいずれかと平行であるものをいう. 三次元レクトリニア多面体は相対位置が固定された直方体の集合として考えられる. 以下では, 与えられる三次元レクトリニア多面体の各々をアイテムと呼ぶ. なお, 座標の x 軸, y 軸, z 軸はそれぞれ容器とアイテムの幅, 高さ, 奥行きに対応し, 左 (同様に下, 奥) に行くほど x (同様に y, z) 座標は小さくなるものとする. 本研究では直方体の容器の最も奥の面の形状 (幅と高さ) は固定されているものとし, 全てのアイテムを詰め込む際に，容器の奥行きが最小になる配置を求める. また, 特に明記しない限り, アイテムの回転は許さないものとする. この問題には, 家具などの荷物の詰め込みや, CNC ルーターを使ったアイテムの切り出しなどの応用がある.

[1] 名古屋大学大学院情報学研究科
[a)] umeda@nagoya-u.jp
[b)] yannanhu@nagoya-u.jp
[c)] yagiura@nagoya-u.jp

本研究では, まず, 二次元レクトリニア図形配置問題に対して提案された構築型解法である bottom-left 法と best-fit 法 [3] を三次元空間に拡張し, 三次元レクトリニア多面体配置問題に対する deepest-bottom-left (DBL) 法と三次元 best-fit (3BF) 法を提案する. DBL 法は, 全てのアイテムに順番が与えられ, その順番にアイテムを一つずつその DBL 点に配置することを繰り返す手法である．アイテムの DBL 点とは，アイテムを容器の中に配置できる位置の中で，最も奥行きの小さい面の最も低い位置のうち, できるだけ左の位置である. 3BF 法は, アイテムに優先順位が与えられているとき, アイテムを一つずつ配置する方法である. 毎回アイテムを配置する際に, 全ての未配置のアイテムの DBL 点を計算し, その内, 最も奥行きが小さい面にある点の中の最も下に位置するものの中で最も左の点を DBL 点として持つアイテムを配置し, 同点の場合には優先順位の最も高いアイテムを配置する. 以上の操作を, 全てのアイテムを配置し終えるまで繰り返す.

また, Hu らが提案した partition-based best-fit 法 [4] を三次元に拡張し, 三次元 partition-based best-fit (3PBF) 法も提案する. 3PBF 法は, 与えられたアイテムをいくつかのグループに分割し, グループごとに 3BF 法を適用する操作を繰り返すことで, すべてのアイテムを配置する手法である. グループ分けを行うことで, DBL 法と 3BF 法の短所を緩和することができる.

グループ分けの方法については, あらかじめアイテムの体積や底面積, 高さなどによって分別しておく方法や, 前回の配置結果をもとに適応的にアイテムの配置順序と分割を

決める方法を提案する.

また, 多面体同士の重なりを容易に判定するために, no-fit cube (NFC) という考え方を提案し, Find2D-BL 法 [2] , 高度なデータ構造, および効率よく NFC を保存する手法を組み込むことにより計算の高速化を行う.

計算実験では, 提案したアルゴリズムの性能を評価し, 問題例によっては計算時間が 1/8 程度に短縮されたことを確認した. さらに, 3PBF 法について, 分割の仕方における挙動の変化を調べ, 基準に体積を用いると比較的良い結果が得られること, および，分割を繰り返し行う方法において，2 回目か 3 回目のグループ分割の際に良い結果を出力する傾向があることを確認した.

2. 三次元レクトリニア多面体配置問題

三次元レクトリニア多面体詰込み問題とは, 与えられた n 個のアイテムの集合 $R = \{R_1, R_2, \ldots, R_n\}$ を幅 W, 高さ H の容器の中に重ならないように全て配置し, 奥行きを最小にする問題である.

容器の幅 W と高さ H が定数として与えられ, 容器の奥行きを変数 D とする. 座標の x 軸, y 軸, z 軸はそれぞれ容器の幅, 高さ, 奥行きに対応し, 左 (同様に下, 奥) に行くほど x (同様に y, z) 座標は小さくなるものとする. ある面が xy 平面に水平になるように配置された直方体の xy 平面に水平な 2 つの面のうち, z 座標の大きい側を前面, 小さい側を背面あるいは底面とよび, その面積を底面積と呼ぶ. アイテム R_i を囲む最小の直方体を bounding box と呼び, その幅, 高さ, 奥行きをそれぞれ w_i, h_i, d_i とする (図 1). また, bounding box の最も奥の面の左下の点を参照点と呼んで, $v_i = (x_i, y_i, z_i)$ によってアイテム R_i の配置位置を表す. すなわち, R_i の参照点が v_i となるように R_i を配置することを, R_i を v_i に配置するという. ミンコフスキー和 $R_i \oplus v_i = \{p + v_i \mid p \in R_i\}$ によってアイテム R_i を位置 v_i に配置した時に R_i が占有する領域を表し, また， R_i の内側の領域を $\mathrm{int}(R_i)$ と表現する. 以上を用いて, 三次元レクトリニア図形配置問題は

$$\begin{aligned}
&\min \quad D && && (1)\\
&\text{s.t.} \quad 0 \le x_i \le W - w_i && (1 \le i \le n), && (2)\\
&\qquad 0 \le y_i \le H - h_i && (1 \le i \le n), && (3)\\
&\qquad 0 \le z_i \le D - d_i && (1 \le i \le n), && (4)\\
&\qquad \mathrm{int}(R_i \oplus v_i) \cap (R_j \oplus v_j) = \emptyset && (i \ne j) && (5)
\end{aligned}$$

と定式化することができる. (2), (3), (4) は容器の中にアイテムを配置することを表し, (5) はアイテムが他のアイテムと重ならないことを表す.

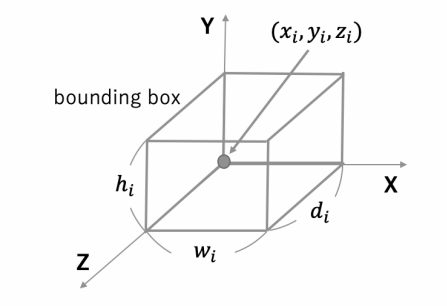

図 1 座標の定義
Fig. 1 Definition of coordinates

3. 準備

3.1 deepest bottom left (DBL) 点

アイテム R_1 から R_{k-1} が既配置であるとき, アイテム R_k がアイテム R_1 から R_{k-1} のいずれとも重なりなく容器の中に置ける位置を配置可能点と呼ぶ. アイテム R_k の DBL 点とは, 配置可能点の中で, 最も奥行きが小さい面上にあるもののうち, 最も下の点の中の, 最も左の点のことである. また奥行きをある定数に固定した xy 平面上に配置可能点があるとき，それらの最も下の点の中で, 最も左の点のことをその平面における BL 点と呼ぶ. このとき, DBL 点は, 配置可能点を含む xy 平面の中で最も奥行きが小さい面上の BL 点に等しい.

3.2 no-fit cube (NFC)

二次元のレクトリニア図形配置問題を解く際, 図形の BL 点を効率よく見つけるための方法として, no-fit polygon (NFP)[1] の考え方が用いられる.

NFP とは, 二次元平面上において 2 つの図形の重なりを判定するのに用いられる幾何学的手法であり, 2 つの図形を重なりなく配置できる領域を描くことができる. 2 つの図形 P と Q が与えられ, P の位置が固定されたとする. Q が P と重るような Q の参照点の集合が P に対する Q の NFP であり, NFP(P, Q) と表す. 例えば, (x_i, y_i) にある幅 w_i, 高さ h_i の長方形 i に対する幅 w_j, 高さ h_j の長方形 j の NFP は,

$$\begin{aligned}
\mathrm{NFP}(i,j) = \{(x,y) \mid\ & x_i - w_j < x < x_i + w_i,\\
& y_i - h_j < y < y_i + h_i\}
\end{aligned} \qquad (6)$$

と計算することができる.

NFP の考え方を三次元空間に拡張し, no-fit cube (NFC) という考え方を定義する.

NFC は 2 つの三次元のアイテムの重なりを判定する. NFP と同様に, NFC は位置が固定された多面体 R と移動できる多面体 S が与えられるとき, S が R と重なるような S の参照点の集合が R に対する S の NFC であり, NFC(R, S) と表される. 例えば, (x_i, y_i, z_i) にある幅 w_i, 高

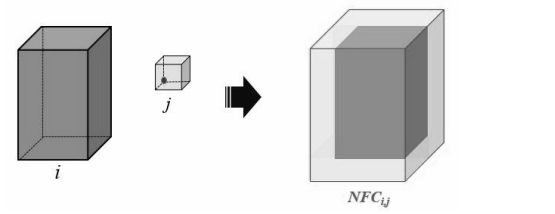

図 2 直方体 i に対する直方体 j の NFC
Fig. 2 NFC of cuboid j for cuboid i

さ h_i, 奥行き d_i の直方体 i に対する幅 w_j, 高さ h_j, 奥行き d_j の直方体 j の NFC は,

$$\begin{aligned}\mathrm{NFC}(i,j) = \{(x,y,z) \mid\ & x_i - w_j < x < x_i + w_i,\\ & y_i - h_j < y < y_i + h_i,\\ & z_i - d_j < z < z_i + d_i\} \end{aligned} \quad (7)$$

と計算することができる (図 2).

三次元レクトリニア多面体 R_i の固定された直方体 R_j に対する $\mathrm{NFC}(R_i, R_j)$ の計算については, アイテム R_i を直方体に分け, 固定された R_j に対して，それぞれの NFC を計算し，得られた直方体の NFC の和集合が $\mathrm{NFC}(R_i, R_j)$ となる.

3.3 容器の表現

$R_1, \ldots, R_n$ のアイテムの奥行きの総和を D_{all} とおき, 幅が W, 高さが $H+2$, 奥行きが $D_{\mathrm{all}}+2$ である直方体 C_1, C_2 を座標 $(-W, -1, -1)$ と $(W, -1, -1)$ に配置する. 同様に, 幅が $W+2$, 高さが H, 奥行きが $D_{\mathrm{all}}+2$ である直方体 C_3, C_4 を座標 $(-1, -H, -1)$ と $(-1, H, -1)$ に, 幅が $W+2$, 高さが $H+2$, 奥行きが 1 である直方体 C_5 を座標 $(-1, -1, -1)$ に配置する.

このように奥，上下左右の 5 つの面のそれぞれに対し，容器の外側からその面に接する大きな直方体 $C_1, \ldots, C_5$ を配置することで, それら 5 つの直方体のいずれにも重複しない位置の集合が容器の内部領域に等しくなるようにする. このように容器を表現することで,「容器の外側にはみ出さない」という制約を,「他のアイテムと重ならない」という制約と同じ扱いで計算できる.

3.4 DBL 点の計算方法

アイテム $R_1, \ldots, R_{k-1}$ が既配置であるとき, アイテム R_k の DBL 点を計算したいとする. まず, 既配置の全てのアイテムに対してアイテム R_k の NFC を計算する. アイテム R_k を m_k 個の直方体 R_k^1, R_k^2, …, $R_k^{m_k}$ で表す. このとき, 既配置のアイテムに対する NFC は

$$\begin{aligned}&\mathrm{NFC}(R_1,R_k^1)\cup\mathrm{NFC}(R_2,R_k^1)\cup\ldots\cup\mathrm{NFC}(R_{k-1},R_k^1)\cup\\ &\mathrm{NFC}(R_1,R_k^2)\cup\mathrm{NFC}(R_2,R_k^2)\cup\ldots\cup\mathrm{NFC}(R_{k-1},R_k^2)\cup\\ &\vdots\\ &\cup\mathrm{NFC}(R_1,R_k^{m_k})\cup\mathrm{NFC}(R_2,R_k^{m_k})\cup\ldots\cup\mathrm{NFC}(R_{k-1},R_k^{m_k})\end{aligned} \quad (8)$$

として計算できる.

次に, 各 NFC の前面の z 座標を昇順に並び替え, その列を $\tilde{d}_1, \tilde{d}_2, \ldots, \tilde{d}_{m_k M_{k-1}} (M_{k-1} = \sum_{i=1}^{k-1} m_i)$ とする. そして, $t = 1, 2, \ldots$ の順に，DBL 点が見つかるまで各 t に対して以下の操作を行う. まず, z 座標が $\tilde{d}_t$ $(1 \le t \le m_k M_{k-1})$ の面において, 内部領域が平面 $z = \tilde{d}_t$ と重複を持つ NFC の各々に対して, その NFC の幅と高さからなる長方形を考える. そのような長方形同士の上辺と右辺の交点を全て求め, 各交点に対して, その点を内部領域に含む長方形を数え上げ, そのような長方形数をその交点の重複度とする. この操作を全ての交点に対して終えた後, 重複度が 0 である交点が存在すれば, それらの内, 最も下にある点の中で, 最も左の点を DBL 点として出力する (そのような交点が存在しないときは t を 1 つ増やした次の反復に移行する).

4. 三次元レクトリニア多面体配置問題を解くアルゴリズム

4.1 deepest-bottom-left (DBL) 法

二次元のレクトリニア図形詰込み問題を解く代表的な手法の 1 つとして, bottom-left (BL) 法が知られている. BL 法とは, 配置すべき図形の詰め込む順番が与えられたときに, それぞれの図形を BL 点に従って配置していく手法である.

deepest-bottom-left (DBL) 法とは, この BL 法を三次元に応用したものであり, 与えられた順序に従ってアイテムを DBL 点に配置していく手法である. アイテムにはあらかじめ順序が与えられており, アイテムの配置はその順序に従って行う. アイテムの順序の決め方にはアイテムの面積や奥行きの降順などが挙げられる. DBL 点にアイテムを配置する点は後述する 3BF 法も同様であるが, アイテムを詰め込む順番が固定されており, 必ずその順番に従ってアイテムを配置することが DBL 法の大きな特徴である.

4.1.1 DBL 法の基本的な実装法

簡単のため, アイテムを詰め込む順序を添え字の昇順とする. また, 計算中の容器の奥行きを MinDepth と表し, その初期値を 0 としておく.

アイテム $R_1, \ldots, R_{k-1}$ が既配置であるとき, アイテム R_k の DBL 点を計算し, アイテム R_k をその DBL 点に配置する. このとき, R_k の DBL 点 (x_k, y_k, z_k) について, $z_k + d_k$ の値を MixDepth と比較し, $z_k + d_k$ の方が大きければ MinDepth を MinDepth := $z_k + d_k$ と更新する.

以上の操作を k の値を 1 つずつ増やして $k = n$ となるま

で繰り返すことで, 全てのアイテムを容器内に配置することができる. また, 全ての操作が終わったとき, MinDepth がその容器の奥行きとなる. 各アイテム R_k の配置位置は, (x_k, y_k, z_k) となる.

4.1.2 DBL 法の計算量

各アイテム R_i を表現するのに要する直方体数 m_i に対して,

$$M = \sum_{i=1}^{n} m_i \tag{9}$$

$$m = \max_i m_i \tag{10}$$

とする. まず, DBL 点の候補として考慮すべき z 座標の値について, この値はいずれかの NFC の前面の z 座標の値に等しいから, その数も NFC の前面の値の種類数に等しい. NFC の数は既配置のアイテムの直方体数の総和と, 今から詰めるアイテムの直方体数の積に等しいから, その総量は $O(mM)$ となる.

次に, 1 つの面に関して, NFC の交点の数は $O(m^2M^2)$ であり, 各交点が他の NFC の内部に含まれているかどうかを調べることは $O(mM)$ 時間で計算できるため, 全ての点を計算するのに必要な計算量は $O(m^3M^3)$ である. よって, 全ての面の交点を調べるのに必要な計算量は $O(m^4M^4)$ であり, 1 つのアイテムを配置するために必要な計算量が $O(m^4M^4)$ であることが分かる.

以上より, DBL 法を用いて n 個のアイテム全てを配置するのに必要な計算量は, $O(nm^4M^4)$ となる.

4.2 三次元 best-fit (3BF) 法

BL 法と同様に, best-fit (BF) 法も二次元のレクトリニア図形詰込み問題を解く代表的な手法として知られている. BF 法とは, 図形に優先順位が与えられているとき, 未配置の全ての図形の BL 点の集合の内, 最も下の点の中で, 最も左の点に, その点に配置できる図形の中で優先順位の最も高いものを配置することを繰り返す手法である.

三次元 best-fit (3BF) 法とは, この BF 法を三次元に応用したものであり, 図形に優先順位が与えられているとき, 未配置の全ての図形の DBL 点の集合の内, 最も奥行きが小さい面の中で, 最も下の点の中の, 最も左の点に, その点に配置できる図形の中で優先順位の最も高いものを配置することを繰り返す手法である.

4.2.1 3BF 法の基本的な実装法

アイテムの優先度は添え字に関係なくあらかじめ定められているとする.

$k-1$ 個のアイテムが既配置であるとする. このとき, DBL 法では次に配置するアイテムは決まっていたが, 3BF 法では残りの全てのアイテムが次に配置されるアイテムの候補となる. 簡単のため, 残りのアイテムを $R_k, \ldots, R_n$ とする. まず, アイテム R_k の DBL 点を DBL 法のときと同様に求める. その後, アイテムは配置せず, その DBL 点を保存しておき, アイテム R_{k+1} の DBL 点を求める. この操作を繰り返し, 残りの全てのアイテムの DBL 点を求める. 全てのアイテムの DBL 点が求まれば, その中で最も奥行きが小さい面のうち, 最も下の点の中で最も左の点を探し, その点を DBL 点に持つアイテムを配置する.

アイテムを配置すれば, DBL 法と同様に MinDepth の値を更新し, 全てのアイテムが配置されたとき, MinDepth がの容器の奥行きとなる.

4.2.2 3BF 法の計算量

DBL 法と同様に, 式 (9), (10) を用いて計算すると, 1 つのアイテムの DBL 点を求めるのに必要な計算量は $O(m^4M^4)$ となる. ここで, n 個のアイテムの内, 形状が異なるものの種類の数を t とすると, 同じ形のアイテムについては DBL 点が同じになるため, 未配置のアイテム全ての DBL 点の計算量が $O(tm^4M^4)$ となり, 1 つのアイテムを配置するのに必要な計算量は $O(tm^4M^4)$ となる.

よって, 3BF 法を用いて n 個のアイテム全てを配置するのに必要な計算量は, $O(ntm^4M^4)$ となる.

4.3 三次元 partition-based best-fit (3BF) 法

Hu らが提案した partition-based best-fit 法は, 与えられたアイテムをいくつかのグループに分割し, グループごとに best-fit 法を用い, 1 つの容器にすべてのアイテムを配置する手法である. これにより, 終盤に影響力の大きいアイテムが残ってしまう BF 法の短所を緩和することができる.

三次元 partition-based best-fit (3PBF) 法とは, この PBF 法を三次元に応用したものであり, あらかじめアイテムを何らかの基準でグループに分割し, 影響力の大きいアイテムを優先して配置しつつ, 3BF 法を繰り返す手法である.

グループの分割方法は大きく分けて 2 つ考えられる.

1 つ目はアイテムを詰める前に何らかの基準を設けてグループ分けをする方法であり, その基準にはアイテムの体積や, bounding box の底面積, 高さなどが考えられる. アイテムの体積を基準にグループ分けする場合には, アイテムを体積の降順に並べ, 隣同士のアイテムとの体積の差が最も大きい部分でアイテムを 2 つのグループに分割し, それを複数回行うことで, 1 つの問題例に対して 2 分割から複数分割までの分割方法を考えることができる. 具体的に, 5 つのアイテム R_1, R_2, R_3, R_4, R_5 の体積がそれぞれ $20, 18, 15, 10, 9$ であるときを考えると, 最も体積の差が大きいのは R_3 と R_4 の間であるから, グループはまず $\{R_1, R_2, R_3\}$ と $\{R_4, R_5\}$ に分割できる (2 分割). さらに分割を続けると, 次に最も体積の差が大きいのは R_2 と R_3 の間であるから, グループはさらに $\{R_1, R_2\}$ と $\{R_3\}$ と $\{R_4, R_5\}$ に分割できる (3 分割). このようにしてできた各分割に対して 3PBF 法を適用し, 最も充填率の高くなる分割方法, 及びアイテムの配置位

置を決定する. なお, この分割を続けると最終的にはアイテムを体積の大きい順に並べた 3DBL 法を行うこととなる.

2 つ目は実際に 1 度アイテムを配置した後で, 各アイテムに対してある基準をもとに並び替えを行い, その基準に基づいてグループの 1 つを分けて新しい分割を作成し, 再度アイテムを配置することを繰り返す方法である. この基準には今回のアイテムの配置位置の深さと 1 つ前の配置位置の深さの差などが挙げられる. 例えば, あるアイテムに対し, 前回は比較的深さの小さい位置に配置できたいたものが, 今回はるかに深さの大きい位置に配置することになってしまった場合には, そのアイテムは影響力の大きいアイテムと判断し, 序盤に配置するアイテムのグループに含まれるように並び替えを行う. 具体的には, 5 つのアイテム R_1, R_2, R_3, R_4, R_5 の 1 回目の配置位置の深さがそれぞれ $0, 1, 3, 2, 5$ で, 2 回目の配置位置の深さが $1, 5, 1, 0, 4$ であった場合, 1 回目と 2 回目の配置位置の深さの差は $-1, 4, -2, 2, -1$ であり, この結果からアイテムを R_2, R_4, R_1, R_5, R_3 と並べ, その差が最も大きい R_4 と R_1 の間でグループを $\{R_2, R_4\}$ と $\{R_1, R_5, R_3\}$ に分割する. その後この分割に 3PBF 法を適用し, 2 回目と 3 回目の配置位置の深さの差をもとに再度グループの分割を繰り返していく. この一連の操作を, 計算結果が変化しなくなるか制限時間を迎えるまで繰り返す.

3PBF 法のアルゴリズムはグループ分けを行う部分以外は 3BF と同様である. また, 計算量については 1 つの問題例に対して各グループごとに 3BF 法を行うことになるが, その計算量が 3BF 法より大きくなることはない.

5. より高速な実現方法

5.1 Find2D-BL 法

Find2D-BL 法は, そのアルゴリズム中で走査線 (sweep-line) の考え方を利用し, 計算方法とデータの保存方法をさらに工夫した手法である.

走査線とは, 容器の底から上に移動する x 軸に平行な線のことであり, この線を移動させる際に容器内の点がどれだけの NFC に含まれているのかを計算することができる. この手法を用いることで, 1 つの面での BL 点を高速に計算することができる. 具体的なアルゴリズムとしては, ある 1 つの面を考えたとき, まず y 座標が 0 である x 軸に平行な走査線について, その走査線がいずれかの NFC の内部を通っているならば, その NFC の左辺から右辺までの x 座標の重複度に 1 を加える. この操作を走査線が内部を通っているすべての NFC に対して行う. その後, 走査線を容器の底から上に x 軸方向に平行に移動させ, NFC の下辺と重なったときにその NFC の左辺から右辺までの x 座標の重複度に 1 を加え, NFC の上辺と重なったときにその NFC の左辺から右辺までの x 座標の重複度から 1 減らす. この操作を繰り返し, いずれかの x 座標の重複度が 0 になったとき, 操作を終了してその点を BL 点として定める.

走査線をそのまま利用することで 1 つの面で BL 点を求めるのに必要な計算時間を $O(m^2M^2)$ に抑えることができる. そのため, この考え方を DBL 法に適用すると, n 個のアイテム全てを配置するのに必要な計算量を $O(nm^4M^4)$ から $O(nm^3M^3)$ に改善することができる. また, より計算方法とデータの保存方法を工夫した Find2D-BL 法は, 1 つの面で BL 点を求めるのに必要な計算時間を $O(mM \log M)$ に抑えることを可能にし, DBL 法の計算時間を $O(nm^2M^2 \log M)$ に, 3BF 法の計算時間を $O(tnm^2M^2 \log M)$ に改善することができる.

5.2 NFC-Layout

三次元レクトリニア多面体を容器に詰め込む際, アイテムの形が全て異なっているという状況は少なく, 同じ種類のアイテムが複数ずつ存在していることが多い. そのようなとき, 全てのアイテムに対して配置するごとに NFC を計算し, DBL 点を求めようとすると, 同じ計算を何度も繰り返すことになり, 余分な計算時間を必要とする. この余分な計算を削減するために, NFC-Layout という考え方を用いる.

NFC-Layout とは, アイテムの種類ごとに NFC の計算結果を保存しておくもので, 対応する種類のアイテムを配置する際に, 既に計算された結果を用いることができる. 例えば, t 種類のアイテムが存在するとすると, 必要となる NFC-Layout は t 個である. ここで, あるアイテム R_i を配置したとき, 全ての t 個の NFC-Layout を更新し, それぞれの DBL 点を計算する. このとき, 1 つの NFC-Layout の 1 つの面に対して, 走査線を用いることで $O(\log M)$ で新たな BL 点を見つけることができ, 最も奥行きが小さい面での BL 点を DBL 点とすることで DBL 点を計算できる. そして, この 1 つの面での計算は最大 mM 回行われるから, 1 つの面の計算量は $O(mM \log M)$ であり, 1 つの NFC-Layout には最大 mM 個の面が存在するから, 1 つの NFC-Layout を構築したのち更新していくのに必要な計算時間は $O(m^2M^2 \log M)$ となる. よって, t 種類の NFC-Layout を構築したのち更新していくのに必要な総計算時間は $O(tm^2M^2 \log M)$ となり, 全ての NFC-Layout を更新し終えたときに全アイテムの配置が終了するから, DBL 法と 3BF 法の計算時間を $O(tm^2M^2 \log M)$ へとさらに改善することができる.

6. 計算実験

DBL 法, 3BF 法, 3PBF 法の 3 つのアルゴリズムを用い, 計算時間と充填率の変化を分析する様々な計算実験を行った. なお, 計算実験は全て MacBook Air (OS X Yosimite, CPU: 1.3 GHz Intel Core i5, メモリ: 4 GB) 上で行った.

6.1 計算量の改善による計算時間の変化の比較

単純な方法で実装した $O(nm^4M^4)$ 時間のプログラム (単純プログラム) と, Find2d-BL 法と NFC-Layout を用いて計算の高速化を行った $O(tm^2M^2 \log M)$ 時間のプログラム (改善プログラム) の計算時間の比較を, DBL 法を用いて行った.

問題例にはアイテム数が 300, 500, 800, 1000, 1500, 2000, 2500, 3000 である 8 種類のものを用意した. 全ての問題例に対して, 容器の幅と高さの値は同じであり, 容器の背面は正方形になるように設定した. また, アイテムについては, bounding box の 1 辺が 10000 以下の二次元平面におけるレクトリニア図形を, 奥行き (z 軸) 方向に伸長させることによって作成した. その際, 奥行きは $1000, 2000, 4000, 8000$ のいずれかの値をランダムにとるものとした. さらに, 容器の幅及び高さは全アイテムの総体積の 1/3 乗とした.

このようにして作った問題例に対する計算結果を表 1 に示す. 計算量のみが変わっており, 配置方法はいずれも DBL 法であるから, 各問題例に対して充填率は変わっていない. 計算時間に関しては改善プログラムの方が全ての問題例に対して速く, その差はアイテム数が大きくなるに連れて開いていく. この差の開きを生み出しているのは, 1 つの面での BL 点を探すのに要する時間の差であると考えられる. なぜなら, アイテム数を増やすと問題例の作成方法から底面の幅と高さが大きくなるため, 各面に重複する NFC の数が増え, 面での操作に工夫を施した改善プログラムの優位性がより大きくなるからである.

6.2 3 つのアルゴリズムの充填率の比較

この実験では, 様々な問題例に対して DBL 法, 3BF 法, 3PBF 法を用いて計算を行い, その充填率を比較する.

この計算実験の 3PBF 法のグループ分けはアイテムを詰める前に行う方法を採用し, アイテムの体積を基準にグループ分けする場合には, アイテムを体積の降順に並べ, 隣同士のアイテムとの体積の差が最も大きい部分でアイテムを 2 つのグループに分割し, それを複数回行うことで, 1 つの問題例に対して 2 分割から複数分割までのグループ分けを行うものとする.

問題例のアイテム数は全て 150 個で, 15 種類の形状のアイテムを 10 個ずつ作ることによって用意した. アイテムは bounding box の 1 辺が 10 以下の二次元平面におけるレクトリニア図形を, 奥行き (z 軸方向) に伸長させることによって作成した. その際, 奥行きは $1, 2, 4, 8$ のいずれかの値をランダムにとるものとした. また, 全ての問題例に対して, 容器の幅と高さの値は同じであり, 容器の背面は正方形になるように設定した. さらに, 容器の幅及び高さは全アイテムの総体積の 1/3 乗とした. このようにして作った問題例の内, 最も体積の大きいアイテムの体積と, 最も体積の小さいアイテムの体積の比がおよそ 10 倍となる問題例を instance1, 比がおよそ 2 倍となる問題例を instance2, 比がおよそ 1.4 倍となる問題例を instance3 とする. また, 各問題例に対して, アイテムの並びとグループ分けの基準を体積にしたものと底面積にしたものの 2 つで計算実験を行った.

このようにして作った問題例に対する計算結果を表 2 に示す. 結果として, アイテムの並びとグループ分けの基準を底面積にしたものは, 基準を体積にしたものよりも充填率が高くなることはなかった.

instance1 に関しては, DBL 法で最も高い充填率を出力できている. これは, アイテムの体積の差が大きい場合には先に体積の大きいアイテムから配置してしまった方が良いということを示しており, たとえ途中でより奥に配置することができる小さなアイテムがあったとしても, 後半に体積の大きいアイテムを残してしまうことの方がデメリットが大きく, 返って結果を悪くしてしまったと考えられる.

instance3 に関しては, 3BF 法で最も高い充填率を出力できている. これは, アイテムの体積の差が小さい場合には, その都度最も奥に詰めることのできるアイテムを優先して配置するべきであることを示している. これは, 体積の近いアイテム同士は充填率に対する影響力にあまり差がなく, 体積によるアイテムの並び替えがほとんど意味を成さない為であると考えられる.

instance2 に関しては, 3PBF 法が最も高い充填率を出力している. この点について, 最も体積の大きいアイテムの体積と, 最も体積の小さいアイテムの体積の比がおよそ 2 倍であるとき, DBL 法が上手くはたらくほど体積に差がないため, その都度配置するアイテムを変えるべきではあるが, 3BF 法ではそこそこ影響力の大きいアイテムを後半に残してしまうこととなり, 影響力を上手く分散させた 3PBF 法が良い結果を出力したと考えられる.

また, これら 3 種のアルゴリズムが出力する解の傾向は, 3PBF 法の分割数ごとの充填率の変遷からも読み取れる. 今回の問題例は 15 種類のアイテムから作成されているため, 体積による分割は最大 15 回行われることになるが, アルゴリズムの性質上, 分割を全く行わない 3PBF 法は 3BF 法と同じ結果を出力し, アイテムを種類ごとに最大数 (今回は 15 回) 分割する 3PBF 法は, DBL 法と同じ結果を出力する. そのため, 3PBF 法は分割数を増やすほど 3BF から DBL 法へとその性質を近づけていくことになると言える.

今回の計算実験における分割数ごとの 3PBF 法の充填率の変遷を図 3 に示す. 図から分かるように, 最も体積の大きいアイテムの体積と, 最も体積の小さいアイテムの体積の比がおよそ 10 倍となる instance1 については, 分割数を増やすほど, つまり DBL 法に近づけば近づくほど充填率が高くなっている. また, その比がおよそ 1.4 倍となる instance3 については, 分割数が少なくなるほど, つまり 3BF 法に近づけば近づくほど充填率が高くなっている. これらの結果は, 上で述べた DBL 法と 3BF 法が上手くはたらくための

表 1 計算量の改善による計算時間の変化
Table 1 The change of computation time by reducing the theoretical computational complexity

問題例	アイテム数	幅×高さ	充填率	計算時間 (sec)	
				単純プログラム	改善プログラム
instance300	300	30000 × 30000	68.72%	0.83	0.63
instance500	500	40000 × 40000	69.56%	4.78	1.76
instance800	800	45000 × 45000	71.86%	17.35	4.88
instance1000	1000	50000 × 50000	75.18%	29.53	7.40
instance1500	1500	56000 × 56000	75.51%	84.18	17.58
instance2000	2000	62000 × 62000	76.73%	173.06	30.77
instance2500	2500	67000 × 67000	77.62%	336.48	48.24
instance3000	3000	71000 × 71000	78.11%	574.99	71.95

表 2 3 つのアルゴリズムの充填率の比較
Table 2 Comparison of packing ratio among three algorithms

問題例	アイテム数	アイテムの種類	グループ分けの基準	充填率		
				DBL 法	3BF 法	3PBF 法
instance1	150	15	体積	71.5%	61.3%	71.5%
instance1	150	15	底面積	61.3%	61.3%	61.3%
instance2	150	15	体積	63.9%	66.3%	68.8%
instance2	150	15	底面積	61.7%	63.9%	63.9%
instance3	150	15	体積	68.1%	73.8%	73.8%
instance3	150	15	底面積	68.1%	73.8%	73.8%

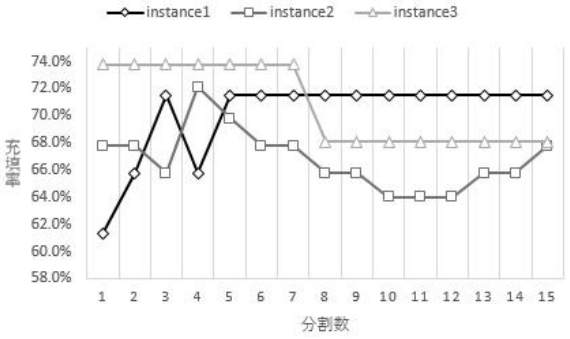

図 3 分割数ごとの 3PBF 法の充填率 (体積基準)
Fig. 3 The packing ratio of the 3PBF algorithm against the number of groups in the partition (volume-based)

問題例の条件に沿うものであると言える.

6.3 配置位置の z 座標による場合分けを行った再帰的な 3PBF 法

6.2 節で扱った 3PBF 法は, グループの分割を問題例に対してあらかじめ行うことで複数の新たな問題例を作成しておき, 各問題例に対して計算を行うものであった. 本節の実験では, 実際に 1 度アイテムを 3PBF 法で配置した後, 各アイテムに対してある基準をもとに並び替えを行い, その基準に基づいてグループを分割しなおし, 再度アイテムを配置することを繰り返す方法を採用する. これは 4.3 節で述べた 3PBF 法の 2 つ目のグループの分割方法である.

この計算実験では, 各アイテムに対する今回のアイテムの配置位置の深さと 1 つ前の配置位置の深さの差を基準として用いる. 具体的には, 5 つのアイテム R_1, R_2, R_3, R_4, R_5 の 1 回目の配置位置の深さが $0, 1, 3, 2, 5$ で, 2 回目の配置位置の深さが $1, 5, 1, 0, 4$ であった場合, 1 回目と 2 回目の配置位置の深さの差は $-1, 4, -2, 2, -1$ であり, この結果からアイテムを R_2, R_4, R_1, R_5, R_3 と並べ, その差が最も大きい R_4 と R_1 の間でグループを $\{R_2, R_4\}$ と $\{R_1, R_5, R_3\}$ に分割する. その後, この分割を用いて 3PBF 法を適用し，その結果とその直前の分割に対する 3PBF 法の結果の配置位置の深さの差をもとに再度グループの分割を繰り返す.

問題例のアイテム数は全て 150 個で, 15 種類の形状のアイテムを 10 個ずつ作ることによって用意した. アイテムは bounding box の 1 辺が 10 以下の二次元平面におけるレクトリニア図形を, 奥行き (z 軸) 方向に伸長させることによって作成した. その際, 奥行きは $1, 2, 4, 8$ のいずれかの値をランダムにとるものとした. なお, 3PBF 法が上手くはたらくように, 6.2 節の結果から, 最も体積の大きいアイテムの体積と最も体積の小さいアイテムの体積の比が 1.5 から 2 になるようにした. また, 全ての問題例に対して, 容器の幅と高さの値は同じであり, 容器の背面は正方形になるように設定した. さらに, 容器の幅及び高さは全アイテムの総体積の 1/3 乗とした. また, 分割回数は 15 回までとし, 1 回目の分割は DBL 法と 3BF 法の配置結果をもとに行うこととした.

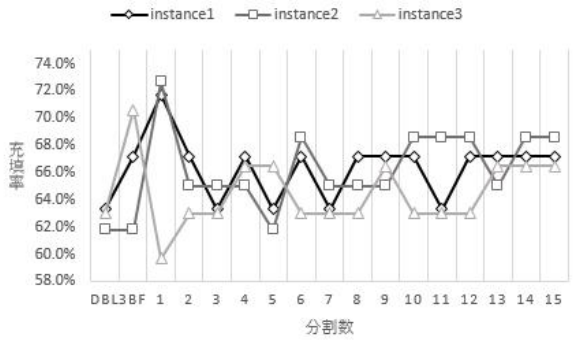

図 4 分割数ごとの 3PBF 法の充填率 (z 座標基準)
Fig. 4 The packing ratio of the 3PBF algorithm against the number of groups in the partition (z coordinate-based)

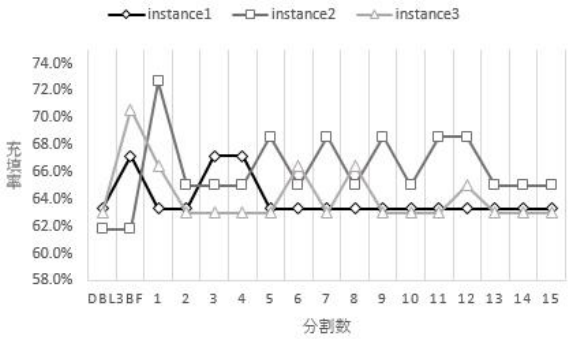

図 5 分割数ごとの 3PBF 法の充填率 (Δz 基準)
Fig. 5 The packing ratio of the 3PBF algorithm against the number of groups in the partition(Δz-based)

今回の計算実験における分割数ごとの 3PBF 法の充填率の変遷を図 4 に示す. 図から分かるように, 今回の問題例に対しては充填率は序盤に大きく変動し, その後分割を行うに連れてその変化は小さくなっている. また, 最も高い充填率や最も低い充填率は 1 回目や 2 回目のように少ない分割数の時に出力され, その後も変動はするものの, それらの値を上回ることはなかった.

6.4 Δz の値による場合分けを行った再帰的な 3PBF 法

この実験では 6.3 節と同様に, 再帰的な 3PBF を用いて行う. この実験で用いる並び替え, および分割に用いる基準はアイテムごとの Δz の値である. アイテムの Δz とは, あるアイテムを配置した後, もう一度その状態で同じアイテムを配置した際の, 配置位置の z 座標の値の差のことである. 例えば, アイテム R_i を深さ 10 の位置に配置したとき, その状態でアイテム R_i の DBL 点を再度計算し, 計算された DBL 点の z 座標が 12 であった場合, アイテム R_i の Δz は 2 となる. この計算方法から分かるように, Δz の値は非負となる. Δz の値は, 局所的ではあるがそのアイテムの影響力を計ることができ, 大きい影響力を持つと考えられる体積や底面積の大きなアイテムは, Δz の値が大きくなりやすく, いつ配置しても結果があまり変わらないような小さなアイテムは, Δz の値が小さくなりやすい.

分割方法には 6.3 節と同じものを用いた. ただし, 6.3 節では 1 回目の分割に 2 つの計算結果が必要だったのに対し, 今回の実験では 1 つの計算結果から 1 回目の分割を行う事ができるため, 3BF 法の計算結果を採用し, DBL 法の計算結果は 3PBF 法に用いていない.

今回の計算実験における分割数ごとの 3PBF 法の充填率の変遷を図 5 に示す. Δz を基準とした 3PBF 法は, どの分割においても平均的な充填率を出力し, DBL 法を下回る充填率が見られることはなかった. しかし, 結果として 6.3 節の手法と比較すると結果は悪くなっている. これは, 問題例によっては同じ形のアイテムであっても, 配置するタイミングによって Δz の値が大きく変わってしまうものもあり, 影響力をうまく計れない場合があることが原因であると考えられる.

7. まとめ

本研究では, 三次元レクトリニア多面体配置問題を解く方法について, DBL 法, 3BF 法, 3PBF 法の 3 つのアルゴリズムを提案し, 実装方法と計算時間について考えた. また, それらを実装するにあたって, より効率的に計算を行うために no-fit cube の考え方を提案し, Find2D-BL 法と NFC-Layout を用いたさらなる計算の高速化を実現した.

計算実験では, 三次元レクトリニア多面体配置問題を解く 3 つのアルゴリズムの性能を評価した. また, 計算の高速化の効果も確認した. さらに, 3PBF 法の分割の仕方における挙動の変化, より効果が発揮されるグループの分割方法を模索し, 得られた結果について, そのような結果が得られた理由を考察した. 結果として, DBL 法と 3BF 法を適用した結果から 6.3 節の分割方法を用いた 3PBF 法を分割数 3 まで行い, それらの結果と 3BF 法の結果の中から最も良い解を選択することが, 充填率, 計算時間を考慮した上で比較的良い戦略であるといえる.

参考文献

[1] R.C. Art Jr.: *An Approach to the Two Dimensional Irregular Cutting Stock Problem*, PhD thesis, Massachusetts Institute of Technology, (1966)
[2] S. Imahori, Y. Chien, Y. Tanaka, M. Yagiura: Enumerating bottom-left stable positions for rectangle placements with overlap, *Journal of the Operations Research Society of Japan*, 57 (2014), 45–61
[3] Y. Hu, H. Hashimoto, S. Imahori, M. Yagiura: Efficient implementations of construction heuristics for the rectilinear block packing problem, *Computers & Operations Research*, 53 (2015), 206–222
[4] Y. Hu, H. Hashimoto, S. Imahori, T. Uno, M. Yagiura: A partition-based heuristic algorithm for the rectilinear block packing problem, *Journal of the Operations Research Society of Japan*, 59 (2016), 110–129