A-017

# 厳密解法と発見的手法の組合せによるサイズ可変ビンパッキング問題の解法

# Solving Methods for the Variable Sized Bin Packing Problem by Combining of Exact Algorithms and Heuristics

三橋 一郎* 大山口 通夫 野呂 耕三†
Ichiro Mitsuhashi Michio Oyamaguchi Kozo Noro

## 概 要

ビンパッキング問題とは，与えられたアイテムをビン（箱）につめる際に使用するビンの数を最小化する問題である．サイズ可変ビンパッキング問題とは，サイズの違うビンが複数種類存在し，すべてのアイテムをつめたときのビンのサイズの総和を最小化する問題である．これらの問題は，建築資材から部材を切り出すときの計画作成など産業的に非常に大きな意味を持つ問題であるが，NP 困難問題であることが知られており，入力の規模が大きいと最適解を計算することが困難になる．理論的な近似精度の解析を伴う近似アルゴリズムがこれまでにいくつか提案されているが，本研究では現実的な入力に対して高精度な解を高速に計算することを目的とし，厳密解法と発見的手法を組み合わせた新しいアルゴリズムを提案する．

## 1 はじめに

ビンパッキング問題とは，与えられたアイテムをビン（箱）につめる際の使用するビンの数を最小化する問題である．サイズ可変ビンパッキング問題とは，サイズの違うビンが複数種類存在し，すべてのアイテムをつめたときのビンのサイズの総和を最小化する問題である．ここで，例えばビンを建築資材（木材，鋼材，管材，棒材など），アイテムを建築物のパーツとなる部材と解釈すると，ビンパッキング問題は資材コストが最小になるような部材の切り出し方を計算する問題にも相当し，産業的にも非常に重要な問題であると言える．

ビンパッキング問題は NP 完全問題であることが知られており，入力の規模が大きいと最適解を現実的な時間内で計算することが困難であるため，いくつかの近似アルゴリズムがこれまでに提案されている [2, 3, 1, 5, 4]．しかしながら，ビンパッキング問題に関しては「P≠NP の仮定のもとで，任意の $\varepsilon > 0$ に対して近似保証 $3/2 - \varepsilon$ をもつアルゴリズムは存在しない」ということが知られている [1]．

本研究はサイズ可変ビンパッキング問題を対象とし，建築現場で必要とされるような現実的な入力[1]に対して高精度な解を高速に計算するアルゴリズムを与えることであるが，前述の事実より近似手法にのみ基づく方法では限界がある．そこで本研究では，厳密解法と発見的手法を組み合わせた新しいアルゴリズムを提案する．まず，本論文の第 3 節で分枝限定法に基づく新しい高速な厳密解法を与える[2]．次に，この厳密解法を用いても現実的な時間内で最適解を求めることが困難な入力に対応するための発見的手法を第 4 節で与える．具体的には，入力されたアイテムを「大アイテム」と「小アイテム」に分割し，大アイテムに対しては第 3 節の手法を用いて最適解を求めた上で，その解に小アイテムを極力最適性を失わないように組込むという手法である．最後に第 5 節では，第 3 節の手法と第 4 節の手法を効果的に組み合わせる方法を与える．

## 2 準備

N を非負整数の集合とする．$\mathrm{N}^n$ を，非負整数を n 個並べた列の集合とする．$\vec{u} = (u_1, \ldots, u_n)$ を非負整数ベクトルとする．$\vec{u}(i) = u_i$ とする．$\mathbf{fni}(\vec{u}) = \mathbf{min}\{i \in \{1, \ldots, n\} \mid \vec{u}(i) \neq 0\}$ とする．$\mathbf{sum}(\vec{u}) = \sum_{k=1}^{n} u_k$ とす

*三重大学総合情報処理センター
†三重大学大学院工学研究科

[1]アイテムとビンの大きさはそれぞれ整数値で与えられ，ビンの大きさは高々6100 程度．アイテムの総数は高々6〜700 程度．

[2]分枝限定法に関する関連研究としては，最大クリーク問題に対するものなどがある [6]

る．$\mathbf{dim}(\vec{u}) = n$ を $\vec{u}$ の次元とする．$\mathbf{dim}(\vec{u}) = \mathbf{dim}(\vec{v})$ のとき，$\vec{u}+\vec{v}$，$\vec{u}-\vec{v}$ を $\vec{u}$ と $\vec{v}$ のそれぞれ和，差とする．すべての $k \in \{1,\ldots,n\}$ について $u(k) \leq v(k)$ が成立するとき，$\vec{u} \leq \vec{v}$ とする．$\vec{0}$ を零ベクトルとする．

$[s_1,\ldots,s_n]$ を多重集合とする．$\sqcup, \setminus$ をそれぞれ多重集合の和，差，$\emptyset$ を空集合とする．

順序集合 $(S, \preceq)$ について，その極大元の集合を $\mathbf{Max}_{\preceq}(S)$ と表記する．

**定義 2.1**

1. $l$ 種類のアイテムの列をアイテムベクトルと呼び，$\vec{a} = (a_1,\ldots,a_l)$ とする（ここで，$a_1 > \cdots > a_l$）．$A = \{a_1,\ldots,a_l\}$ とする．$k \in \{1,\ldots,l\}$ について，$\mathbf{ind}(a_k) = k$ とする．$l$ 次元ベクトル $\vec{u} = (u_1,\ldots,u_l) \in \mathbb{N}^l$ に対して $\mathbf{isum}(\vec{u}) = \sum_{k=1}^{l} a_k \cdot u_k$ とする．この $\vec{u}$ を数量ベクトルと呼ぶ．

2. $r$ 種類のビンの列をビンベクトルと呼び，$\vec{b} = (b_1,\ldots,b_r)$ とする（ここで，$b_1 > \cdots > b_r$）．$B = \{b_1,\ldots,b_r\}$ とする．$\mathbf{price} : B \to \mathbb{N}$ を，各ビンのコストを返す単調関数とする．

3. $\mathbf{isum}(\vec{p}) \leq \vec{b}(1)$ を満たす数量ベクトル $\vec{p}$ をパックと呼ぶ．$\mathbf{tightbin}(\vec{p}) = \mathbf{min}\{b \in B \mid b \geq \mathbf{isum}(\vec{p})\}$ とする．$\mathbf{expansion}(\vec{p}) = (\overbrace{\vec{a}(1),\ldots,\vec{a}(1)}^{\vec{p}(1)}, \overbrace{\vec{a}(2),\ldots,\vec{a}(2)}^{\vec{p}(2)}, \ldots, \overbrace{\vec{a}(l),\ldots,\vec{a}(l)}^{\vec{p}(l)})$ とする．

4. アイテムベクトル $\vec{u}$ について，次の条件を満たすパック $\vec{p}$ の集合を $P_{\vec{u}}$ とする：$\vec{p} \leq \vec{u}$．

**定義 2.2** パックの多重集合 $P = [\vec{p_1},\ldots,\vec{p_n}]$ $(\vec{p_1},\ldots,\vec{p_n} \in P_{\vec{u}})$ について，

1. $\mathbf{cost}(P) = \sum_{k=1}^{n} \mathbf{price}(\mathbf{tightbin}(\vec{p_k}))$．
2. $\sum_{k=1}^{n} \vec{p_k} = \vec{u}$ が成立するとき，P は $\vec{u}$ における充足パック集合であるという．
3. $\vec{u}$ における充足パック集合 $P$ のうち，$\mathbf{cost}(P)$ が最小のものを $\vec{u}$ に対する最適パック集合と呼ぶ．

コスト一般化サイズ可変ビンパッキング問題の入出力は以下の通りである．

入力
- アイテムベクトル $\vec{a}$ と $\mathbf{dim}(\vec{a}) = \mathbf{dim}(\vec{u})$ を満たすベクトル $\vec{u}$
- ビンベクトル $\vec{b}$ とコスト関数 $\mathbf{price} : B \to \mathbb{N}$

出力 $\vec{u}$ に対する最適パック集合

**例 2.3** 入力の一例を以下に示す．

- $\vec{a} = (3646, 3576, 1820)$
- $\vec{u} = (1, 1, 2)$
- $\vec{b} = (6096, 3048)$
- すべての $b \in B$ について $\mathbf{price}(b) = b$

# 3 厳密解法

この節ではまず，与えられた入力に対する最適パック集合を計算する厳密解法を与える．

## 3.1 初期解の構築

数量ベクトル $\vec{w}$ が与えられたとき，その入力に対してある程度最適解に近い充足パック集合 $S$ を計算する発見的アルゴリズムをグリーディ法で構築する．

```
function Greedy(w⃗)
  S := ∅;
  while |w⃗| ≠ 0⃗
    p⃗ := hcp(P_w⃗);
    S := S ⊔ [p⃗];
    w⃗ := w⃗ − p⃗
  return S
```

ここで，$\mathbf{hcp}(P_{\vec{w}})$ は，$P_{\vec{w}}$ の中から $\mathbf{price}(\mathbf{tightbin}(\vec{p}))/\mathbf{isum}(\vec{p})$ の値が最小になるパック $\vec{p}$ を一つ返す関数である．このような関数 **hcp** は，ナップサック問題の動的計画法による解法などを用いて高速なものを実現することができる．**Greedy** は，**hcp** によって計算された $\vec{p}$ を $S$ に追加し，その都度 $\vec{w}$ から $\vec{p}$ を減算し，$\vec{w} = \vec{0}$ となるまで繰り返すだけの単純なアルゴリズムである．

## 3.2 分枝限定法

数量ベクトル $\vec{w}$ が与えられたとき，その最適パックリスト $O$ を分枝限定法によって求めるアルゴリズム **OPT** を以下に示す．

```
procedure OPT(w⃗)
  O := Greedy(w⃗);
  BB(∅, w⃗)

procedure BB(S, w⃗)
  if w⃗ = 0⃗ then
    if cost(S) < cost(O) then
      O := S
  else
    for each p⃗ ∈ P_w⃗
          such that fni(p⃗) = fni(w⃗)
      if cost(S ⊔ [p⃗]) < cost(O) then
        BB(S ⊔ [p⃗], w⃗ − p⃗)
```

$O$ に **Greedy** で求めた $\vec{u}$ における充足パック集合を初期解として格納することで，アルゴリズム中の枝刈り（下から 2 行目の **if** 文）を効果的に行うことができる．手続き **BB** の **for each** 文において $\mathbf{fni}(\vec{p}) = \mathbf{fni}(\vec{w})$ という条件が入っているが，深さ優先探索木の各頂点においては $\vec{w}$ の「0 でない最左の要素のインデックス」に着目し，そのインデックスの値が一致するパック $\vec{p}$ のみを頂点として探索を続けるという意味である．これはすなわち，探索木の各頂点において，残存するアイテムの中で一番大きいものを含むパックのみを頂点とするということに相当する．例えば，例 2.3 の入力にアルゴリズム **OPT** を適用すると，最初に呼び出された **BB** の **for each** 文における $\vec{p}$ の取りうる値は，$(1,0,1),(1,0,0)$ の 2 通り（**expansion** 適用形式で記述すると $(3646,1820),(3646)$ の 2 通り）である．それぞれに対して再帰呼び出しを実行すると，次の $\vec{p}$ の取りうる値は，$(0,1,1),(0,1,0)$ の 2 通り（**expansion** 適用形式で記述すると $(3576,1820),(3576)$ の 2 通り）である．このように探索範囲を制限することで，最適性を失うことなく探索効率を上げることができる．最終的に，$O$ には $\vec{u}$ に対する最適パック集合が格納される．

本アルゴリズムは再帰関数を使って記述されており，呼び出されるたびに集合 $P_{\vec{w}}$ を再計算しているが，これをスタックと **while** ループを用いたプログラムに変換し，$P_{\vec{w}}$ もあらかじめすべて計算しておいてテーブルに格納しておけば，実装レベルでの高速化を図ることができる．

## 3.3 改良分枝限定法

コスト関数と **fni** の値による枝刈りを行う分枝限定法アルゴリズム **OPT** を与えたが，これでも最適解を求める上で必要のない探索を多く行っているため，まだ高速化の余地が残されていると言える．ここでは，「パック集合の極大元」に着目して，アルゴリズムの高速化を図る．

**定義 3.1**

1. 半順序関係 $\preceq \subseteq \mathbb{N}^+ \times \mathbb{N}^+$ を次の通り定義する：$(c_1,\ldots,c_m),(d_1,\ldots,d_n) \in \mathbb{N}^+$ について，$m \leq n$ を満たし，かつすべての $k \in \{1,\ldots,m\}$ について $c_k \leq d_k$ が成立するとき，$(c_1,\ldots,c_m) \preceq (d_1,\ldots,d_n)$ とする．

2. 半順序関係 $\preceq \subseteq P_{\vec{u}} \times P_{\vec{u}}$ を次のとおり定義する：パック $\vec{p_1},\vec{p_2}$ について，$\mathbf{tightbin}(\vec{p_1}) = \mathbf{tightbin}(\vec{p_2})$ かつ $\mathbf{expansion}(\vec{p_1}) \preceq \mathbf{expansion}(\vec{p_2})$ が成立するとき，$\vec{p_1} \preceq \vec{p_2}$ とする．

**OPT** を高速化した **OPTR** を以下に示す．

```
procedure OPTR(w⃗)
  O := Greedy(w⃗);
  BBR(∅, w⃗)

procedure BBR(S, w⃗)
  if w⃗ = 0⃗ then
    if cost(S) < cost(O) then
      O := S
  else
    for each p⃗ ∈ Max_⪯(P_w⃗)
          such that fni(p⃗) = fni(w⃗)
      if cost(S ⊔ [p⃗]) < cost(O) then
        BBR(S ⊔ [p⃗], w⃗ − p⃗)
```

アルゴリズム **OPT** との相違点として，手続き **BB** では **for each** 文の $\vec{p}$ を $P_{\vec{w}}$ から選択していたが，手続き **BBR** では極大元の集合 $\mathbf{Max}_{\preceq}(P_{\vec{w}})$ に制限されている．例えば，例 2.3 の入力にアルゴリズム **OPT** を適用したときの最初に呼び出される **BB** の **for each** 文における $\vec{p}$ の取りうる値（**expansion** 適用形式で記述する）は，$(1,0,1),(1,0,0)$ の 2 通り（**expansion** 適用形式で記述すると $(3646,1820),(3646)$ の 2 通り）であったが，**BBR** においては $(1,0,1)$ の 1 通り（**expansion** 適用形式で記述すると $(3646,1820)$）に削減される．さらに再帰呼び出しを実行したときの次の $\vec{p}$ の取りうる値は，$(0,1,1)$ の 1 通り（**expansion** 適用形式で記述すると $(3576,1820)$）であり，最終的に $O$ に格納されるのは $(1,0,1),(0,1,1)$（**expansion** 適用形式で記述すると $(3646,1820),(3576,1820)$）となる．このような変更を行っても，アルゴリズム終了時の $O$ には $\vec{u}$ に対する最適パック集合が格納されていることを以下に示す．

**定理 3.2** アルゴリズム **OPTR** 終了時の $O$ は，$\vec{w}$ に対する最適パック集合である．

**証明** $\vec{w}$（ただし，関係 $\preceq$）に関する帰納法で証明する．$\vec{w}$ に対する最適パック集合を $\mathcal{O}_{\vec{w}}$ とする．$\mathcal{O}_{\vec{w}}$ の要素として最初に選んだパックを $\vec{q} \in P_{\vec{w}}$ とする．$\vec{q} \in \mathbf{Max}_{\preceq}(P_{\vec{w}})$ の場合，帰納法の仮定より $\vec{w}-\vec{q}$ に対して，**OPTR** は最適パック集合を求めることができる．したがって，**OPTR** は $\vec{w}$ に対しても最適パック集合を求めることができる．$\vec{q} \notin \mathbf{Max}_{\preceq}(P_{\vec{w}})$ の場合，$\vec{p} \geq \vec{q}$ を満たす $\vec{p} \in \mathbf{Max}_{\preceq}(P_{\vec{w}})$ が存在する．$\vec{w}-\vec{q} \geq \vec{w}-\vec{p}$ が成立するので，$\mathbf{cost}(\mathcal{O}_{\vec{w}-\vec{q}}) \geq \mathbf{cost}(\mathcal{O}_{\vec{w}-\vec{p}})$．ここで，$\mathbf{cost}(\mathcal{O}_{\vec{w}-\vec{q}}) > \mathbf{cost}(\mathcal{O}_{\vec{w}-\vec{p}})$ と仮定すると，$\mathbf{cost}(\mathcal{O}_{\vec{w}}) = \mathbf{price}(\mathbf{tightbin}(\vec{q})) + \mathbf{cost}(\mathcal{O}_{\vec{w}-\vec{q}}) > \mathbf{price}(\mathbf{tightbin}(\vec{p})) + \mathbf{cost}(\mathcal{O}_{\vec{w}-\vec{p}})$ となり，$\mathcal{O}_{\vec{w}}$ の最適性に矛盾する．よって，$\mathbf{cost}(\mathcal{O}_{\vec{w}-\vec{q}}) = \mathbf{cost}(\mathcal{O}_{\vec{w}-\vec{p}})$ が成立する．したがって，$\mathbf{cost}(\mathcal{O}_{\vec{w}}) = \mathbf{price}(\mathbf{tightbin}(\vec{q})) + \mathbf{cost}(\mathcal{O}_{\vec{w}-\vec{q}}) = \mathbf{price}(\mathbf{tightbin}(\vec{p})) + \mathbf{cost}(\mathcal{O}_{\vec{w}-\vec{p}})$ が成立する．帰納法の仮定より，$\mathbf{cost}(\mathcal{O}_{\vec{w}-\vec{p}})$ は **OPTR** で求めることができるので，したがって，**OPTR** は $\vec{w}$ に対しても最適パック集合を求めることができる． □

いくつかのサンプルデータを用いて **OPT** と **OPTR** の性能比較を行った結果を表 1 に示す．

# 4 組込み法

入力のサイズが大きく，前節で述べた改良分枝限定法アルゴリズムを用いても最適解を求めることが困難である場合が存在する．本節では，入力されたアイテムを大アイテムと小アイテムに分割し，大アイテムに対する最適解を求めた上で，その解に小アイテムを極力最適性を失わずに組込む発見的手法を与える．

## 4.1 アイテムの分割

ここでは，大アイテムと小アイテムに分割する際の指標を与える．明らかに，大アイテムを狭く取りすぎると最終的な解の精度が低くなり，逆に広く取りすぎると現実的な時間内で計算が終了しないことがある．したがって，何らかの根拠に基づいた指標を与える必要がある．

**定義 4.1**

1. 数量ベクトル $\vec{u}$ と $I \subseteq \{1,\ldots,\mathbf{dim}(\vec{u})\}$ について，$\vec{u}$ の $i(\in I)$ 番目の要素のみを取り出したベクトルを $\vec{u}_{|I}$ とする．すなわち，$\vec{u}_{|I} = (u'_1,\ldots,u'_{\mathbf{dim}(\vec{u})})$，ただし，$i \in I$ のとき $u'_i = \vec{u}(i)$，$i \notin I$ のとき $u'_i = 0$．
2. $$\mathbf{treesize}(\vec{u},d) = \prod_{k=1}^{d} n_k \mathrm{H}_{\vec{u}(k)}$$
ここで，$\vec{u}(k) = 0$ のとき $n_k = 1$，$\vec{u}(k) > 0$ のとき $n_k = |\{\vec{p} \in \mathbf{Max}_{\preceq}(P_{\vec{u}}) \mid \mathbf{fni}(\vec{p}) = k\}|$ とする．

すべての $k \in \{1,\ldots,d\}$ に対して $\{\vec{p} \in \mathbf{Max}_{\preceq}(P_{\vec{u}}) \mid \mathbf{fni}(\vec{p}) = k\}$ から重複を許して $\vec{u}(k)$ 個を取り出したときの組合せ数を計算し，それらをすべて掛け合わせたものが $\mathbf{treesize}(\vec{u},d)$ である．すなわち，アルゴリズム **BBR** における探索木の葉の総数を粗く見積もったものである．したがって，閾値 $t$ を設定した上で $\mathbf{treesize}(\vec{u},d) \leq t$ を満たす最大の $d$ を求め，$\vec{a}(1),\ldots,\vec{a}(d)$ を「大アイテム」，$\vec{a}(d+1),\ldots,\vec{a}(\mathbf{dim}(\vec{a}))$ を「小アイテム」とすることで，精度と計算時間を両立できるような分割が可能となる．このようにして求めた $d$ を $\mathbf{border}(\vec{u},t)$ と表記する．$t$ の値は $10^{16}$ 程度が妥当と考えられるが，これはマシンパワー，許容される計算時間などの要因により変わりうる．

## 4.2 組込み法

ここでは，前節で求めた大アイテムに対する最適パック集合に対して、極力その最適性を失わずに小アイテムを組み込むアルゴリズムを与える．

**定義 4.2**

1. $\vec{p} \in P_{\vec{u}}$ について，$\mathbf{loss}(\vec{p}) = \mathbf{tightbin}(\vec{p}) - \mathbf{isum}(\vec{p})$．これは，パック $\vec{p}$ に対して最も無駄なく使用できるビンのサイズから，$\vec{p}$ 中のアイテムの総和を引いた余りを意味する．
2. $\vec{p_1},\vec{p_2} \in P_{\vec{u}}$ について，$\mathbf{increase}(\vec{p_1},\vec{p_2}) = \mathbf{price}(\mathbf{tightbin}(\vec{p_1}+\vec{p_2})) - \mathbf{price}(\mathbf{tightbin}(\vec{p_1}))$．
3. パックの多重集合 $O$ とパックの集合 $C$ について，下記の条件を満たす $\vec{p_1} \in O, \vec{p_2} \in C$ が存在するとき，$\mathbf{bestcouple}(O,C,\vec{w}) = \langle\vec{p_1},\vec{p_2}\rangle$．下記条件を見たす $\vec{p_1},\vec{p_2}$ が複数通りある場合は，その中の一つを返すものとする．存在しないときは $\mathbf{bestcouple}(O,C,\vec{w}) = \langle\vec{0},\vec{0}\rangle$ とする．
   (a) $\vec{p_1}+\vec{p_2} \in P_{\vec{w}}$
   (b) $\mathbf{loss}(\vec{p_1}+\vec{p_2}) \leq \mathbf{loss}(\vec{p_1})$
   (c) すべての $\vec{p'_1} \in O, \vec{p'_2} \in C$ について，$\mathbf{increase}(\vec{p_1},\vec{p_2}) \leq \mathbf{increase}(\vec{p'_1},\vec{p'_2})$

表 1: **OPT** と **OPTR** の計算時間の比較

| No. | dim | sum | $\lvert P_{\vec{v}}\rvert$ | **OPT** による計算時間 (sec) | $\lvert \mathbf{Max}_{\preceq}(P_{\vec{v}})\rvert$ | **OPTR** による計算時間 (sec) |
|---|---|---|---|---|---|---|
| 1 | 11 | 83 | 89 | 600 超 | 48 | 6 |
| 2 | 16 | 197 | 108 | 600 超 | 38 | 600 超 |
| 3 | 6 | 161 | 17 | 252 | 9 | 1 未満 |
| 4 | 5 | 73 | 23 | 89 | 20 | 6 |
| 5 | 7 | 57 | 67 | 600 超 | 40 | 59 |

関数 **bestcouple** の定義の条件 (b) は，$\vec{p_1}$ に対して $\vec{p_2}$ を組み込んだとき，もとの $\vec{p_1}$ より余りが大きくならないということを意味している．(使用するビンは大きくなっても良い)

アルゴリズム **Fit** を次のとおり定義する．

**procedure Fit**$(O, \vec{w}, n)$
  $i := 1$;
  **while** $i \leq \mathbf{dim}(\vec{w})$
    $C := \{\vec{p} \in P_{\vec{w}} \mid \mathbf{sum}(\vec{p}) \leq n, \mathbf{fni}(\vec{p}) = i\}$;
    $\langle \vec{p_1}, \vec{p_2} \rangle := \mathbf{bestcouple}(O, C, \vec{w})$;
    **if** $\vec{p_1} = \vec{0}$ **then** //(**bestcouple** が存在しない)
      $i := i + 1$
    **else**
      $O := O \setminus [\vec{p_1}] \sqcup [\vec{p_1} + \vec{p_2}]$;
      $\vec{w} := \vec{w} - \vec{p_2}$

**Fit**$(O, \vec{u}, n)$ を実行すると，パックの多重集合 $O$ に $\{\vec{p} \in P_{\vec{u}} \mid \mathbf{sum}(\vec{p}) \leq n\}$ 中のパックが組み込まれ，$O$ に再格納される．つまり，パラメータ $n$ は一度に組み込めるアイテムの数の上限を意味している．$\vec{u}$ の各要素の値は，$O$ に組み込まれた小アイテムの分だけ減少する．関数 **bestcouple** の定義より，このアルゴリズムでは必ずしもすべての小アイテムが組み込まれるわけではない．

**Fit** は発見的手法によるアルゴリズムであるが，いくつかの条件を満たすときには最適性が保存されるということを以下に示す．

**定理 4.3** $O$ を $\sum_{\vec{p} \in O} \vec{p}$ における最適パック集合，$\vec{p_1} \in O$ に対して $\vec{p_1} + \vec{p_2}$ をパックとする．そのとき，ビンベクトル $\vec{b}$ とパック $\vec{p_2}$ が次の条件を満たすならば，$O \setminus [\vec{p_1}] \sqcup [\vec{p_1} + \vec{p_2}]$ は $\sum_{\vec{p} \in O} \vec{p} + \vec{p_2}$ における最適パック集合である．

1. $\vec{b}$ の条件
   (a) すべての $k \in \{1, \ldots, \mathbf{dim}(\vec{b})\}$ について，$\vec{b}(k) = k \cdot \vec{b}(1)$
   (b) すべての $k, k' \in \{1, \ldots, \mathbf{dim}(\vec{b})\}$ について，$\mathbf{price}(\vec{b}(k))/\vec{b}(k) = \mathbf{price}(\vec{b}(k'))/\vec{b}(k')$
2. $\vec{p_2}$ の条件
   (a) $\mathbf{sum}(\vec{p_2}) = 1$
   (b) $\mathbf{loss}(\vec{p_1} + \vec{p_2}) \leq \mathbf{loss}(\vec{p_1})$

**証明** $O \setminus [\vec{p_1}] \sqcup [\vec{p_1} + \vec{p_2}]$ は $\sum_{\vec{p} \in O} \vec{p} + \vec{p_2}$ における最適パック集合でないと逆に仮定すると，$\mathbf{cost}(O') < \mathbf{cost}(O \setminus [\vec{p_1}] \sqcup [\vec{p_1} + \vec{p_2}])$ を満たす $\sum_{\vec{p} \in O} \vec{p} + \vec{p_2}$ における充足パック集合 $O'$ が存在する．条件 2.(a) より，$\vec{q} > \vec{p_2}$ を満たす $\vec{q} \in O'$ が存在する．条件 1.(a) より，$\mathbf{tightbin}(\vec{q}) - \mathbf{tightbin}(\vec{q} - \vec{p_2}) \geq \mathbf{tightbin}(\vec{p_2}) - \vec{b}(\mathbf{dim}(\vec{b}))$ および $\mathbf{tightbin}(\vec{p_1} + \vec{p_2}) - \mathbf{tightbin}(\vec{p_1}) \geq \mathbf{tightbin}(\vec{p_2}) - \vec{b}(\mathbf{dim}(\vec{b}))$ が成立する．ここで，条件 2.(b) より，$\mathbf{tightbin}(\vec{p_1} + \vec{p_2}) - \mathbf{tightbin}(\vec{p_1}) = \mathbf{tightbin}(\vec{p_2}) - \vec{b}(\mathbf{dim}(\vec{b}))$．よって，$\mathbf{tightbin}(\vec{q}) - \mathbf{tightbin}(\vec{q} - \vec{p_2}) \geq \mathbf{tightbin}(\vec{p_1} + \vec{p_2}) - \mathbf{tightbin}(\vec{p_1})$ が成立するので，条件 1.(b) より，$\mathbf{increase}(\vec{q} - \vec{p_2}, \vec{p_2}) \geq \mathbf{increase}(\vec{p_1}, \vec{p_2})$．したがって，$\mathbf{cost}(O' \setminus [\vec{q}] \sqcup [\vec{q} - \vec{p_2}]) < \mathbf{cost}(O)$ が成立し，$O$ の最適性に矛盾する．よって，命題が成立する． □

この定理は，ビンベクトル $\vec{b}$ が特定の条件を満たしているとき，**Fit**$(O, \vec{u}, 1)$ の実行によって最適性が失われることはないということを意味している．したがって，**Fit**$(O, \vec{u}, n)$ を実行する場合には，まず $n = 1$ として，次に $n = x(> 1)$ として，というように 2 段階に分けることが効果的であると言える．

## 4.3 最終組込み法

ここでは，アルゴリズム **Fit** で組み込めなかった小アイテムを最終的にすべて組み込むアルゴリズムを与える．そのアルゴリズム **FinalFit** を次のとおり定義する．

**procedure FinalFit**$(O, \vec{w})$
  **while** $\vec{w} \neq \vec{0}$
    $\vec{p} := \mathbf{hcp}(P_{\vec{w}})$;
    $\vec{w} := \vec{w} - \vec{p}$;
    $\langle \vec{p_1}, \vec{p} \rangle := \mathbf{bestcouple}(O, \{\vec{p}\}, \vec{w})$;
    **if** $\vec{p_1} = \vec{0}$ **or**

**increase**($\vec{p_1}, \vec{p}$) > **price**(**tightbin**($\vec{p}$)) **then**
$O := O \sqcup [\vec{p}]$
**else**
$O := O \setminus [\vec{p_1}] \sqcup [\vec{p_1} + \vec{p}]$

関数 **hcp** は，アルゴリズム **Greedy** のものと共通である．関数 **hcp** でコストパフォーマンスの良いパックを一つ求め，それを $O$ 中のパックに組み込むか，あるいは独立したパックとして $O$ に加えるということを，$\vec{w} = \vec{0}$ となるまで繰り返す．

## 5 アルゴリズムの流れ

この節では，前節までに与えた厳密解法と発見的手法の効果的な組み合わせ方について述べる．

### 5.1 基本の流れ

1. $d$ := **border**($\vec{u}, t$)，$\vec{u_1}$ := $\vec{u}_{|\{1,\ldots,d\}}$，$\vec{u_2}$ := $\vec{u}_{|\{d+1,\ldots,\mathbf{dim}(\vec{u})\}}$ を実行．(4.1 節)
2. **OPTR**($\vec{u_1}$) を実行し，$O$ を計算．(3.3 節)
3. **Fit**($O, \vec{u_2}, 1$) を実行し，$O$ を再計算．(4.2 節)
4. **Fit**($O, \vec{u_2}, x$) を実行し，$O$ を再計算．(4.2 節)
5. **FinalFit**($O, \vec{u_2}$) を実行し，$O$ を再計算．(4.3 節)

まず，入力されたアイテムを大アイテムと小アイテムに分割し (1)，大アイテムに対する最適パック集合を求める (2)．次に，アルゴリズム **Fit** を用いて小アイテムの組込みを行うが，まずは $n = 1$ としてアルゴリズムを実行する (3)．次に $n = x$（$x$ は 3 から 10 程度が適当と考えられる）としてもう一度 **Fit** を適用し (4)，最後に残った小アイテムを **FinalFit** で組み込む (5)．

### 5.2 反復法

前述のアルゴリズムは，厳密解法と発見的な組込み法を併用することで大きなサイズの入力に対しても高い精度の解を出力できるというものであったが，さらにサイズが大きくなるとやはりあまり良い解を出すことができない．このアルゴリズムにおいては，大アイテムに対する最適解を求めた上で小アイテムを後で追加するという操作を行っているが，一般に入力サイズが大きくなるにつれて最適解を求められる範囲の比率が低くなり，出力される結果が最適解から大きく離れてしまう，ということが原因として挙げられる．この欠点を補うための手法「反復法」を以下に示す．

1. *Solution* := ()，$d$ := **border**($\vec{u}, t$) を実行．(4.1 節)
2. $\vec{u_1} := \vec{u}_{|\{1,\ldots,d\}}$，$\vec{u_2} := \vec{u}_{|\{d+1,\ldots,\mathbf{dim}(\vec{u})\}}$ を実行．
3. **OPTR**($\vec{u_1}$) を実行し，$O$ を計算．(3.3 節)
4. **Fit**($O, \vec{u_2}, 1$) を実行し，$O$ を再計算．(4.2 節)
5. $d' := d$，$\vec{u} := \vec{u_2}$ を実行．
6. すべての $\vec{p} \in O$ に対して，**tightbin**($\vec{p}$) = $\vec{b}(1)$ かつ **loss**($\vec{p}$)/**tightbin**($\vec{p}$) $\leq 0.01$ ならば *Solution* := *Solution* $\sqcup \{\vec{p}\}$，そうでなければ $\vec{u} := \vec{u} + \vec{p}$ を実行．
7. $d$ := **border**($\vec{u}, t$) を計算．(4.1 節)
8. $d \neq d'$ なら 2. にジャンプ．
9. すべての $\vec{p} \in O$ に対して，**tightbin**($\vec{p}$) $\neq \vec{b}(1)$ または **loss**($\vec{p}$)/**tightbin**($\vec{p}$) $> 0.01$ ならば *Solution* := *Solution* $\sqcup \{p\}$ および $\vec{u} := \vec{u} - \vec{p}$ を実行．
10. **Fit**(*Solution*, $\vec{u}, x$) を実行し，*Solution* を再計算．(4.2 節)
11. **FinalFit**(*Solution*, $\vec{u}$) を実行し，*Solution* を再計算．(4.3 節)

このアルゴリズムでは，改良分枝限定法と組込み法 ($n = 1$) を実行した後，得られた結果 $O$ の中から，「最大のビンを使用していて，かつ **loss** が 1% 以下」のパックのみを解として確定し多重集合 *Solution* に格納，それ以外のものは破棄して $\vec{u}$ に戻す．*Solution* に採用されたパックが存在する場合，この $\vec{u}$ はアルゴリズム開始時点の $\vec{u}$ より数量が減っているため，再度分割を行うと大きいアイテムに分類されるアイテムの種類が増え，より広い範囲に対して最適解が求められるようになることがある．この操作を何回か繰り返した後に，組込み法 ($n = x$) と最終組込み法を適用して，最終的な解とする．

### 5.3 アルゴリズムの性能実験結果

アルゴリズム（反復法）の性能実験を行った結果を表 2 に示す．この実験では，実際に建設現場で用いられた実績のあるデータを使っており，またすべてのビン $b$ について **price**($b$) = $b$ としている．表中で（最適解）と記載されているものは，Step 1 で $d = \mathbf{dim}(\vec{u})$ となり（入力されたすべてのアイテムが大アイテムに分類されたこと

表 2: アルゴリズム（反復法）の性能実験結果

| No. | dim | sum | isum | 5.2 節のアルゴリズムで得られる解の総コスト | 総コスト/isum |
|---|---|---|---|---|---|
| 1 | 10 | 83 | 249,036 | 259,080 | 1.04 |
| 2 | 51 | 606 | 573,256 | 573,452 | 1.00 |
| 3 | 16 | 197 | 710,579 | 735,177 | 1.03 |
| 4 | 6 | 161 | 688,343 | 735,177（最適解） | 1.07 |
| 5 | 5 | 73 | 234,951 | 237,744 | 1.01 |
| 6 | 7 | 57 | 209,678 | 217,627 | 1.04 |
| 7 | 12 | 91 | 259,555 | 263,956 | 1.02 |
| 8 | 5 | 72 | 268,224 | 275,539（最適解） | 1.03 |
| 9 | 10 | 94 | 248,565 | 251,155 | 1.01 |
| 10 | 9 | 76 | 235,906 | 244,449 | 1.04 |
| 11 | 7 | 65 | 224,010 | 231,038 | 1.03 |
| 12 | 7 | 78 | 223,491 | 229,819 | 1.03 |
| 13 | 8 | 125 | 462,250 | 496,214 | 1.07 |
| 14 | 7 | 79 | 292,954 | 296,265 | 1.01 |
| 15 | 6 | 91 | 269,696 | 277,368 | 1.03 |
| 16 | 5 | 86 | 305,190 | 315,772 | 1.03 |
| 17 | 9 | 64 | 205,390 | 212,140 | 1.03 |
| 18 | 10 | 63 | 212,145 | 224,332 | 1.06 |
| 19 | 8 | 66 | 232,289 | 240,792 | 1.04 |
| 20 | 8 | 68 | 218,896 | 227,380 | 1.04 |
| 21 | 5 | 56 | 217,112 | 217,627（最適解） | 1.00 |
| 22 | 3 | 58 | 164,444 | 165,201（最適解） | 1.00 |
| 23 | 8 | 66 | 232,744 | 236,524（最適解） | 1.02 |
| 24 | 5 | 72 | 246,063 | 249,936（最適解） | 1.02 |
| 25 | 16 | 118 | 144,625 | 145,694 | 1.01 |
| 26 | 19 | 65 | 187,968 | 193,243 | 1.03 |
| 27 | 53 | 228 | 398,951 | 401,116 | 1.01 |
| 28 | 20 | 72 | 161,159 | 162,763 | 1.01 |
| 29 | 90 | 330 | 475,782 | 477,316 | 1.00 |
| 30 | 67 | 434 | 117,501 | 117,652 | 1.00 |

を意味する），アルゴリズム **OPTR** で最適解を求めることができたことを意味する．**isum** の値を最適解の下界と見なすと，かなり高い近似精度が得られていることがわかる．なお，これらの解はすべて数秒〜数分で求められている．

## 6 まとめ

本研究では，サイズ可変ビンパッキング問題に対して，分枝限定法に基づく効率の良い厳密解法を提案し，また，その厳密解法と発見的な組込み法を組み合わせた高速かつ高精度なアルゴリズムを提案した．今後の課題としては，動的計画法などの手法を用いた厳密解法の更なる高速化や，組込み法のさらなる高精度化が挙げられる．

## 謝辞

この研究は，（株）鈴工との共同研究である．

## 参考文献

[1] L. Epstein and A. Levin. An aptas for generalized cost variable-sized bin packing. *SIAM J. Comput*, 38(1):411–428, 2008.

[2] D. K. Friesen and M. A. Langston. Variable sized bin packing. *SIAM J. Comput*, 15(1):222–230, 1986.

[3] J. Kang and S. Park. Algorithms for the variable sized bin packing problem. *Europian Journal of Operational Research*, 147:365–372, 2003.

[4] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proc. 23rd Annual IEEE SFCS*, pages 312–320, 1982.

[5] F. D. Murgolo. An efficient approximation scheme for variable-sized bin packing. *SIAM J. Comput*, 16(1):149–161, 1987.

[6] Y. Sutani, T. Higashi, E. Tomita, S. Takahashi, and N. Nakatani. A faster branch-and-bound algorithm for finding a maximum clique. *Tech. Rep. IPSJ SIG*, 2006.