

# 修士論文

## 建築パーツ切り出しのための可変サイズ ビンパッキングアルゴリズムについて

平成22年度  
三重大学大学院工学研究科  
計算機ソフトウェア研究室

野呂耕三

# 目次

概要	1
第1章 建築パーツ切り出し問題	2
第2章 可変サイズビンパッキング問題	5
2.1 可変サイズビンパッキング問題の諸定義	5
2.2 可変サイズビンパッキング問題の近似アルゴリズム	6
第3章 探索範囲縮小法	8
3.1 分枝限定法	8
3.2 探索範囲縮小法	10
第4章 ハイブリッド法	12
4.1 動的計画法	12
4.2 ハイブリッド法	13
第5章 高精度近似アルゴリズム	15
5.1 漸近的多項式時間近似スキーム	15
5.2 高精度近似アルゴリズム	15
第6章 実験結果	18
6.1 高速最適化アルゴリズムの評価	18
6.2 高精度近似アルゴリズムの評価	18
6.3 まとめ	18
おわりに	20
参考文献	20
謝辞	22

# 概要

住宅等を建設する際に用いられる柱（以下、**建築パーツ**）は通常、木材から切り出されて出荷されており、業者は必要な**建築パーツ**のリストを受け取って木材から切り出しを行っている。このとき、切り出し方によっては木材を無駄に消費してしまうことがあり、建設コスト、環境負荷の観点から改善が必要とされている。本研究ではこのコスト最適化問題を**建築パーツ切り出し問題**と呼び、効率の良い切り出し方を計算するアルゴリズムを提案する。

建築パーツ切り出し問題は組み合わせ最適化問題の**可変サイズビンパッキング問題**と等価である。この問題はNP 困難問題であることが知られており、入力サイズが大きくなると現実的な時間で最適解を計算することが不可能になる性質がある。しかし、建築パーツ切り出し問題のような現実の問題では比較的短時間で最適化可能な入力サイズが与えられる場合があるため、最適化アルゴリズムを高速化して、入力サイズの許容範囲を広げるとは有益である。そこで、本研究では分枝限定法を高速化した**探索範囲縮小法**と、分枝限定法と動的計画法を併用した**ハイブリッド法**から構成される高速最適化アルゴリズムを提案する。

一方、最適化が困難な場合は、近似アルゴリズムが用いられるが、固定サイズビンパッキング問題に対して近似保証  $3/2$  未満のアルゴリズムは存在しないことが知られているため、理論的に有効な近似アルゴリズムは確立されておらず、実用化されているアルゴリズムも十分な近似精度を達成しているとはいえないのが現状である。そこで、本研究では、固定サイズビンパッキング問題に対する**漸近的多項式時間近似スキーム**の考え方を応用した**高精度近似アルゴリズム**を提案する。

また、本研究では実用化を考慮して一般的なパーソナルコンピュータ上で従来手法と提案手法を実装し、実際に建築パーツを切り出す際に用いられた入力に対し、近似精度の比較を行った。

# 第1章 建築パーツ切り出し問題

住宅等を建設する際に用いられる柱（以下、**建築パーツ**）は一般的に次のように出荷されている。まず、建設業者は必要な建築パーツを記載したリストを加工業者に送信する。そして、加工業者は受け取ったリストに記載されている全ての建築パーツを、工場に用意されている木材から切り出して建設業者へ出荷している。（図 1.1）

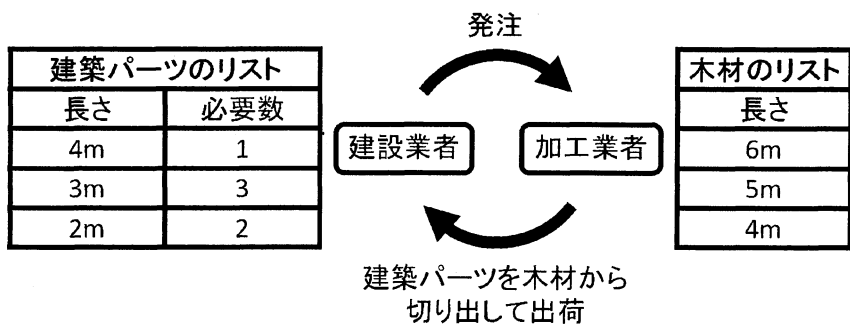


図 1.1: 建築パーツの出荷の流れ

木材の単位長さあたりのコストがすべて等しいとき、建築パーツの切り出しにかかったコストは使用した木材の長さの総和となる。なお、ここでは各木材は十分に用意されていると仮定し、どの木材を使用したかということはコストに影響しないものとする。図 1.2 は建築パーツの切り出しの例である。この例から分かるように、切り出し方によっては木材を無駄に消費してしまう場合がある。

使用木材	切り出す建築パーツ		
6m	4m	2m	
6m	3m	3m	
5m	3m	2m	
使用した木材の長さの総和 = 17m			

使用木材	切り出す建築パーツ		
4m	4m		
6m	3m	3m	
4m	3m		
4m	2m	2m	
使用した木材の長さの総和 = 18m			

図 1.2: 建築パーツの切り出し例

本研究では、上記のような「木材の長さのリスト、建築パーツの長さと必要数のリストが与えられたとき、使用した木材の長さの総和が最小になる切り出し方を求める問題」を**建築パーツ切り出し問題**と呼ぶ。

本研究では実用化を考慮し、株式会社鈴工で実際に使用された入力データを使用する。図 1.3 は提供されたデータの一例である。切り出し方は、木材の長さ、建築パーツのそれぞれの長さにもよるが、一般的に建築パーツの種類数と総数に依存する。図 1.3 の例では、種類数が 16、総数が 118 の入力データであるが、この程度の入力サイズになると、切り出し方が膨大になり、すべての切り出し方を比較して最適解を得るという手法は時間的に不可能になる。したがって、実際には何らかの基準に従って切り出し方を求めるアルゴリズムが用いられている。

図 1.4 は図 1.3 の入力データに対してグリーディー法で計算した結果である。このアルゴリズムは木材の余りが最も少ない切り出し方を選んでいく手法だが、最初に効率の良い切り出し方を求めるため、後半で余りが多くなる切り出し方が計算されることがあり、最適性は保証していない。この他にも様々なアルゴリズムが提案されているが最適解を常に得る多項式時間アルゴリズムは  $P \neq NP$  のもとでは存在しないということが知られている。

アルゴリズムの性能の指標に近似度があり、本研究では (使用した木材の長さの総和/最適解で使用した木材の長さの総和) と定める。したがって、近似度は 1 に近いほど優れていることになるが、最適解が得られない場合は近似度を (使用した木材の長さの総和/建築パーツの長さの総和) とする。たとえば、図 1.4 の切り出し方の近似度は  $149336/144625 \approx 1.032$  で最適解からの誤差は高々 3.2% であると判断する。

木材のリスト		建築パーツのリスト	
2438mm		長さ	必要数
3048mm		2493mm	2
3657mm		2341mm	13
4267mm		2251mm	14
4876mm		2201mm	12
5486mm		2106mm	10
6096mm		2038mm	2
		1659mm	5
		1355mm	1
		1004mm	3
		855mm	1
		749mm	3
		515mm	2
		396mm	3
		286mm	5
		196mm	12
		146mm	30

建築パーツの入力サイズ	
種類数	16
総数	118
長さの総和	144625mm

図 1.3: 実際に使用された入力データ

使用数	使用木材	切り出す建築パーツ												
3	2438mm	146mm	146mm	146mm	146mm	146mm	146mm	146mm	146mm	146mm	196mm	196mm	196mm	286mm
1	2438mm	146mm	146mm	146mm	146mm	146mm	146mm	196mm	196mm	196mm	286mm	515mm	515mm	396mm
1	3657mm	196mm	1355mm	2106mm										
1	4267mm	749mm	855mm	1004mm	1659mm									
1	6096mm	146mm	749mm	749mm	2201mm	2251mm								
1	6096mm	286mm	1659mm	2038mm	2106mm									
1	4267mm	1004mm	1004mm	2251mm										
3	6096mm	1659mm	2201mm	2201mm										
1	4267mm	2038mm	2201mm											
2	4876mm	2341mm	2493mm											
4	4267mm	2106mm	2106mm											
11	2438mm	2341mm												
12	2438mm	2251mm												
4	2438mm	2201mm												
使用した木材の長さの総和 = 149336mm														

図 1.4: 図 1.3 に対する切り出しの例

図 1.5、1.6 は提供されたデータ 50 例のデータ別の建築パーツの種類数、総数を示したグラフである。建築パーツの種類数が 30 未満のデータが全体の約 75%、建築パーツの総数が 200 未満のデータが全体の約 70% を占めている。なお、木材のデータはすべて、図 1.3 に示したデータを用いている。本研究ではアルゴリズムの性能比較にこ 50 件の入力データを実験データとして用いた。

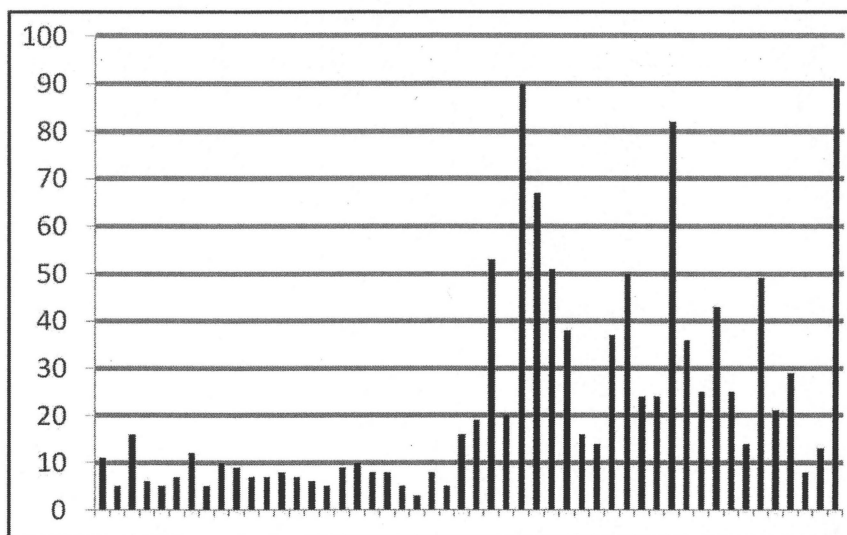


図 1.5: 提供された 50 件のデータ別の建築パーツの種類数

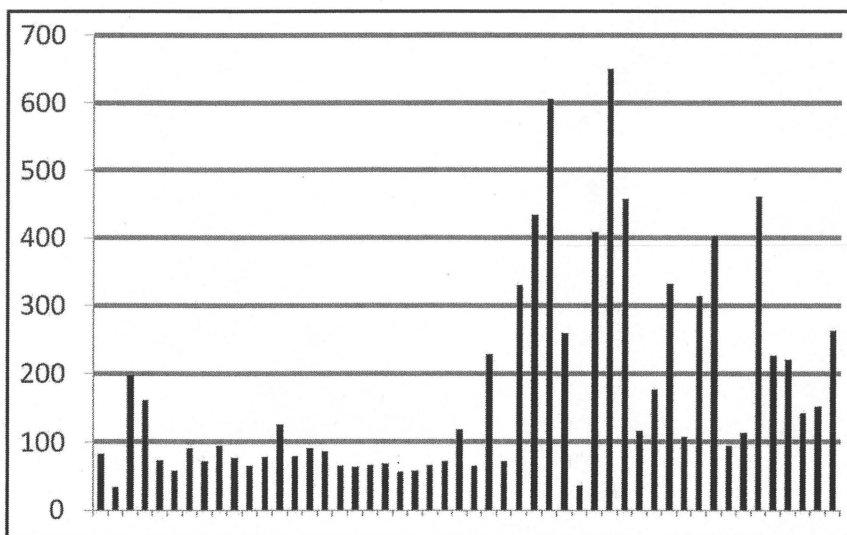


図 1.6: 提供された 50 件のデータ別の建築パーツの総数

## 第2章 可変サイズビンパッキング問題

ここでは、建築パーツ切り出し問題と等価な可変サイズビンパッキング問題について述べる。この問題は、直感的には使用した容器（以下、ビン）の容量の総和が最小になるように荷物（以下、ピース）を詰める問題である。ただし、どの種類のビンも無限個用意されており、単位容量あたりのコストはすべて等しいものとする。ここで、ビンを木材、ピースを建築パーツと考えると、この問題は建築パーツ切り出し問題と等価であることが分かる。つまり、ピースをビンに詰めることは建築パーツを木材から切り出すことに対応している。したがって、可変サイズビンパッキング問題のアルゴリズムは容易に建築パーツ切り出し問題に適用できるため、以後の議論は可変サイズビンパッキング問題の定義に従うものとする。

### 2.1 可変サイズビンパッキング問題の諸定義

#### 定義 2.1 (ビンとピース)

ビンの集合を  $B = \{b_1, b_2, \dots, b_m\}$ 、ピースの集合を  $P = \{p_1, p_2, \dots, p_n\}$  と表し、 $\text{len}()$ 、 $\text{num}()$  はそれぞれビンまたはピースの長さ、個数を表す関数とし、 $|B| = m$ 、 $|P| = n$  と定める。このとき、ビンとピースは  $\text{len}(b_1) \geq \text{len}(b_2) \geq \dots \geq \text{len}(b_m)$ 、 $\text{len}(p_1) \geq \text{len}(p_2) \geq \dots \geq \text{len}(p_n)$ 、 $\text{len}(b_1) \geq \text{len}(p_1)$ 、 $\text{num}(b_i) = \infty$  ( $1 \leq i \leq m$ ) を満たしているものとする。(図 2.1)

#### 定義 2.2 (パターン)

パターンを  $c = (b, p_{q_1} p_{q_2} \dots p_{q_m})$  と表し、 $\text{bin}(c) = \text{len}(b)$ 、 $\text{piece}(c)_i = \text{len}(p_{q_i})$  ( $1 \leq i \leq m$ )、 $\text{pseq}(c) = p_{q_1} p_{q_2} \dots p_{q_m}$ 、 $|c| = m$ 、さらに  $\text{count}(\omega)_i$  をピース列  $\omega$  に含まれる  $p_i$  ( $1 \leq i \leq |P|$ ) の数を表す関数と定める。このとき、 $\text{piece}(c)_1 \geq \text{piece}(c)_2 \geq \dots \geq \text{piece}(c)_m$ 、であり、 $b$  は  $\sum_{i=1}^m \text{piece}(c)_i \leq \text{bin}(c)$  を満たす最小のビンとする。また、すべての  $i \in \{1, \dots, |P|\}$  について  $\text{count}(\text{pseq}(c))_i \leq \text{num}(p_i)$  であり、これらを満たす全てのパターンの集合を  $C = \{c_1, c_2, \dots, c_n\}$  と表し、 $|C| = n$  とする。(図 2.2)

#### 定義 2.3 (部分解と解)

部分解を  $\alpha = c_{q_1} \dots c_{q_m}$  と表し、 $\text{cost}(\alpha) = \sum_{i=1}^m \text{bin}(c_{q_i})$ 、 $|\alpha| = m$  と定める。ここで、 $\alpha$  はすべての  $i \in \{1, \dots, |P|\}$  について  $\sum_{j=1}^m \text{count}(c_{q_j})_i \leq \text{num}(p_i)$  を満たしている必要がある。特に、すべての  $i \in \{1, \dots, |P|\}$  について  $\sum_{j=1}^m \text{count}(c_{q_j})_i = \text{num}(p_i)$  が成り立つとき  $\alpha$  を解という。

#### 定義 2.4 (入出力)

入力を  $B$ 、 $P$ 、 $\text{len}()$ 、 $\text{num}()$ 、出力をすべての解の中でコストが最小である解  $\alpha^*$  とする。

#### 定義 2.5 (近似度)

解  $\alpha$  の近似度を  $\text{cost}(\alpha)/\text{cost}(\alpha^*)$  と定める。ただし、 $\alpha^*$  を得るのが困難な場合は  $\text{cost}(\alpha)/\sum_{i=1}^{|P|} \text{len}(p_i) * \text{num}(p_i)$  とする。

B	len(b)	P	len(p)	num(p)
b1	900	p1	510	3
b2	600	p2	470	1
b3	300	p3	320	4
		p4	240	3
		p5	130	3

図 2.1: ビンとピースの例

## 2.2 可変サイズビンパッキング問題の近似アルゴリズム

可変サイズビンパッキング問題はNP 困難問題であり、 $P \neq NP$ のもとでは最適解を得る多項式時間は存在しないため、近似アルゴリズムの研究がこれまでに成されてきた。単純なアルゴリズムの例に First-Fit のアルゴリズムがある。

### アルゴリズム 2.6 (First-Fit)

1. ピースを任意に選ぶ
2. いままでにピースを詰めたビンと並べる
3. 並べた順に試しながら、最初に入るビンにピースを詰める
4. どれにも入らないときは新しいビンにピースを詰める
5. ピースが残っていれば 1 に戻る

このアルゴリズムは  $|B| = 1$  のとき近似保証 2 を達成することを容易に証明することができる。なお、 $|B| = 1$  の問題を可変サイズビンパッキング問題の特別な形として扱い、固定サイズビンパッキング問題、または単にビンパッキング問題という。First-Fit の他にも、さまざまなアルゴリズムが研究されており、たとえば、文献 [1] で提案されている FFDLR 法と FFDLS 法の近似保証はそれぞれ、 $(3/2)OPT+1$ 、 $(4/3)OPT+3$  である。このように理論的な近似精度の解析の研究が行われているが、固定サイズビンパッキング問題では、近似保証  $3/2$  未満のアルゴリズムは存在しないということが知られている。(文献 [2]) これは、可変サイズビンパッキング問題に対しても同じことがいえるため、実用的に有効な近似保証を持つアルゴリズムは事実上、存在しないことになる。(FFDLS 法の近似保証が  $(4/3)OPT+3$  であることは、近似保証  $3/2$  未満のアルゴリズムは存在しないという事実と矛盾しているように見えるが、定数 3 があるため近似精度  $3/2$  以上になる場合があるため、矛盾は生じない)

そこで近似保証のない発見的アルゴリズムがしばしば用いられることがあり、グリーディー法などがその例である。これらのアルゴリズムは経験的または直感的に良いと思われる手法で実用的に広く使用されている。ただし、近似精度にばらつきが出たり、近似精度そのものが悪いことがあるため、万能な手法が確立されているとはいえないのが現状である。



C	bin(c)	piece(c)1	piece(c)2	piece(c)3	piece(c)4	piece(c)5
c1	900	240	240	130	130	130
c2	900	240	130	130	130	
c3	600	130	130	130		
c4	300	130	130			
c5	300	130				
c6	900	320	130	130	130	
c7	900	470	130	130	130	
c8	900	470	130	130		
c9	600	240	130	130		
c10	600	240	130			
c11	900	320	240	130	130	
c12	900	240	240	130	130	
c13	900	240	240	130		
c14	900	510	130	130	130	
c15	900	510	130	130		
c16	900	510	130			
c17	900	320	320	130	130	
c18	900	320	240	130		
c19	900	240	240	240	130	
c20	900	240	240	240		
c21	900	510	240	130		
c22	900	470	240	130		
c23	900	470	240			
c24	600	470	130			
c25	600	320	130			
c26	600	320				
c27	600	240	240			
c28	600	320	130	130		
c29	900	320	240	240		
c30	600	320	240			
c31	900	320	320	240		
c32	900	320	320	130		
c33	900	320	320			
c34	300	240				
c35	900	510	240			
c36	900	470	320			
c37	900	510	320			
c38	600	470				
c39	600	510				

図 2.2: パターンの例

## 第3章 探索範囲縮小法

最適化アルゴリズムは、2.2節で述べた近似アルゴリズムとは対照的に常に最適解を計算する手法である。このアルゴリズムは、可変サイズビンパッキング問題のようなNP困難問題を計算する場合、入力サイズが大きくなると現実的な時間内で計算することは不可能になる性質がある。しかし、建築パーツ切り出し問題のような現実の問題では比較的短時間で計算可能な入力サイズが与えられる場合があるため、最適化アルゴリズムを高速化し、入力サイズの許容範囲を広げることが有益である。ここでは、最適化アルゴリズムの分枝限定法と、分枝限定法を高速化した探索範囲縮小法について述べる。

### 3.1 分枝限定法

最適化アルゴリズムの最も単純な方法に、すべての解を探索する方法がある。このとき、最適解になりえない解の探索を早い段階で打ち切ることで高速化する手法を分枝限定法と呼ぶ。まず、すべての解を列挙するために、探索木を定義する。

#### 定義 3.1 (探索木)

探索木のノードをパターン  $C = \{c_1, c_2, \dots, c_m\}$  とする。このとき、兄弟のノードは添え字の昇順に配置し、 $c_i (1 \leq i \leq m)$  の子ノードは  $c_j (i \leq j \leq m)$  にする。なお、探索の順序はポストオーダーとする。

#### 定義 3.2 (ノードの計算状況)

探索木のあるノード  $c$  の計算状況を、根からノード  $c$  までにたどったパターンの列  $\gamma$  とする。

#### 定義 3.3 (探索条件)

探索木のあるノード  $c$  を探索するのは、ノード  $c$  の計算状況  $\gamma$  が部分解または解のときのみとする。

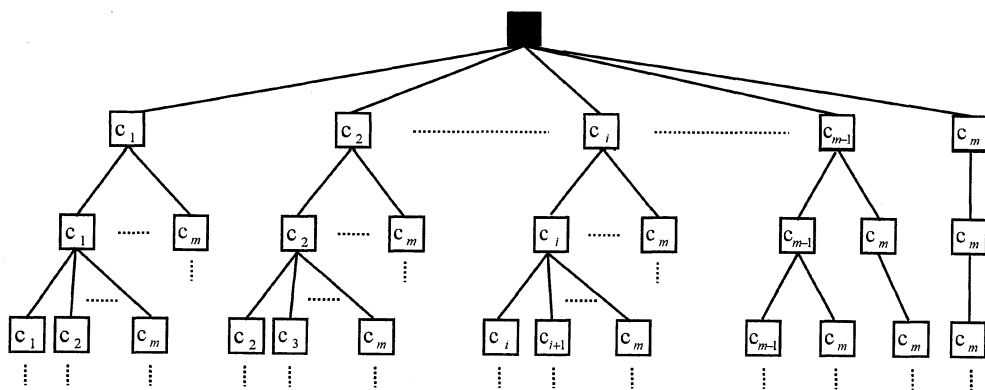


図 3.1: パターンをノードとした探索木

定義 3.1 で定めた探索木を図式化すると図 3.1 のようになる。定義 3.3 にしたがってこの探索木たどると、すべての解を過不足なく列挙することができる。このとき分枝限定法では、 $\alpha^*$  以外の解の探索をなるべく早い段階で打ち切るために、枝刈りと呼ばれる手法が用いられる。以下に、枝刈りに必要な下界値と上界値の定義を示す。

#### 定義 3.4 (下界値)

あるノード  $c$  において、ノード  $c$  の計算状況が  $\gamma = c_{q_1} \cdots c_{q_m}$  のとき、ビンに入れていないピースの集合を  $\mathbf{P}' = \{p'_1, p'_2, \dots, p'_{|\mathbf{P}'|}\}$  とする。つまり、すべての  $i \in \{1, \dots, |\mathbf{P}'|\}$  について  $\text{num}(p'_i) = \text{num}(p_i) - \sum_{j=1}^m \text{count}(c_{q_j})_i$  を満たしている。このとき、ノード  $c$  における下界値を  $L = \text{cost}(\gamma) + \sum_{i=1}^{|\mathbf{P}'|} \text{len}(p_i) * \text{num}(p'_i)$  と定める。

#### 定義 3.5 (上界値)

現在までに得られた解の中で、コストが最小の解を  $\alpha$  としたとき、上界値を  $U = \text{cost}(\alpha)$  と定める。

定義 3.4、3.5 から、あるノードにおける下界値が上界値以上であるとき、そのノードの子孫を探索しても上界値を改善する解は得られないのは自明である。したがって、以下の定理が成立する。

#### 定理 3.6 (枝刈り)

あるノード  $c$  における下界値  $L$  と上界値  $U$  に  $L \geq U$  が成り立つとき、ノード  $c$  の探索は省略可能である。

例えば、図 2.1 から図 2.2 のパターンを得て、探索木を構成したところノード数は 422643 であったが、定理 3.6 の枝刈りを用いた分枝限定法で探索したノード数は 118039 であった。このように、分枝限定法では探索べきノード数を少なくして高速化を図っている。

## 3.2 探索範囲縮小法

本研究では分枝限定法をさらに高速化するために、包含関係とパターンクラス概念を導入し、余分な探索を省略する条件を導く。以下にこれらの定義を与える。

### 定義 3.7 (包含関係)

任意のパターン  $c_i, c_j \in C$  について、 $\text{bin}(c_i) = \text{bin}(c_j)$ 、 $\text{pnum}(c_i) \geq \text{pnum}(c_j)$ 、およびすべての  $k \in \{1, \dots, \text{pnum}(c_j)\}$  について  $\text{piece}(c_i)_k \geq \text{piece}(c_j)_k$  が成立するとき  $c_i$  は  $c_j$  を包含すると定める。

### 定義 3.8 (パターンクラス)

探索木における任意の兄弟のノードの集合を  $\{c_i, c_{i+1}, \dots, c_{|C|}\} (1 \leq i \leq |C|)$  とする。このとき、すべての  $j \in \{m, \dots, n-1\} (i \leq m \leq n-1 < |C|)$  で  $c_j$  が  $c_{j+1}$  を包含するとき、 $\{c_m, c_{m+1}, \dots, c_n\}$  をパターンクラスと定める。

定義 3.7、3.8 から以下の探索範囲縮小の定理が導くことができる。

### 定理 3.9 (探索範囲縮小)

パターンクラス  $\{c_m, c_{m+1}, \dots, c_n\} (1 \leq m < n \leq |C|)$  があるとき、 $c_i (m \leq i < n)$  が探索可能ならば、 $c_j (i < j \leq n)$  の探索は省略可能である。

#### 証明

パターンクラス  $\{c_m, c_{m+1}, \dots, c_n\} (1 \leq m < n \leq |C|)$  の  $c_i (m \leq i < n)$  が探索可能なとき、 $c_j (i < j \leq n)$  から得られるコストが最小の解を  $\alpha^+$  とする。ここで、 $c_j$  の子孫のノードに含まれるピースを  $c_j$  に追加、または  $c_j$  のピースと交換し  $\text{pseq}(c_i) = \text{pseq}(c_j)$  となるように  $c_j$  を  $c_j^*$  に変更し、変更後の解を  $\alpha$  としたとき、 $c_i$  の子孫で  $\text{cost}(\alpha^+)$  以下の解が存在することを示して証明する。

まず、 $c_j$  の子孫のノード  $c$  に含まれるピースを  $c_j$  に追加して  $c'$ 、 $c'_j$  を得る場合を考える。 $c$  からピースを取り出すので  $\text{bin}(c') \leq \text{bin}(c)$  は明らかである。一方、 $c_i$  が  $c_j$  を包含していることから、 $\text{pseq}(c_j)$  が  $\text{pseq}(c_i)$  と等しくなるように変更するときに  $\text{bin}(c'_j) = \text{bin}(c_j)$  が成り立つ。

次に、 $c_j$  の子孫のノード  $c$  のピースを  $c_j$  のノードと交換して  $c'$ 、 $c'_j$  を得る場合を考える。 $c$  の交換するピースを  $p_q$ 、 $c_i$  の交換するピースを  $p_r$  とすると、 $c_i$  は  $c_j$  を包含しているから、 $\text{len}(p_r) \leq \text{len}(p_q)$  が成り立つ。したがって、 $c$  のピース  $p_q$  を  $p_q$  以下の長さのピース  $p_r$  と交換するので  $\text{bin}(c') \leq \text{bin}(c)$  は明らかである。また、同様に  $\text{bin}(c'_j) = \text{bin}(c_j)$  が成り立つ。

以上の議論から、 $c_j$  の子孫のノードに含まれるピースを  $c_j$  に追加、または  $c_j$  のピースと交換するとき、コストは単調減少するため、変更後の解  $\alpha$  は  $\text{cost}(\alpha) \leq \text{cost}(\alpha^+)$  を満たす。一方、 $c_j^*$  は  $c_i$  に等しいので、 $\alpha$  は  $c_i$  の子孫で得ることができる。したがって、 $c_i$  の子孫で  $\text{cost}(\alpha^+)$  以下の解が得られるため、 $c_j$  の探索は省略可能である。□

図 3.2 は図 2.2 のパターンをパターンクラスに分類した例である。定理 3.9 からパターンクラスに含まれるノードの探索は高々 1 つで十分であるから、パターンクラスは実質ひとつのパターンとみなすことができる。この場合、パターン数が 39 でパターンクラス数が 23 であるから、探索木における兄弟のノード数を実質 16 減らしていることになる。実際に定理 3.9 の条件を使用すると探索ノード数は 23894 になり、分枝限定法の探索ノード数 118039 を改善している。

パターンクラス	C	bin(c)	piece(c)1	piece(c)2	piece(c)3	piece(c)4	piece(c)5
1	c1	900	240	240	130	130	130
	c2	900	240	130	130	130	
2	c3	600	130	130	130		
3	c4	300	130	130			
	c5	300	130				
4	c6	900	320	130	130	130	
5	c7	900	470	130	130	130	
	c8	900	470	130	130		
6	c9	600	240	130	130		
	c10	600	240	130			
7	c11	900	320	240	130	130	
	c12	900	240	240	130	130	
	c13	900	240	240	130		
8	c14	900	510	130	130	130	
	c15	900	510	130	130		
	c16	900	510	130			
9	c17	900	320	320	130	130	
	c18	900	320	240	130		
10	c19	900	240	240	240	130	
	c20	900	240	240	240		
11	c21	900	510	240	130		
	c22	900	470	240	130		
	c23	900	470	240			
12	c24	600	470	130			
	c25	600	320	130			
	c26	600	320				
13	c27	600	240	240			
14	c28	600	320	130	130		
15	c29	900	320	240	240		
16	c30	600	320	240			
17	c31	900	320	320	240		
	c32	900	320	320	130		
	c33	900	320	320			
18	c34	300	240				
19	c35	900	510	240			
20	c36	900	470	320			
21	c37	900	510	320			
22	c38	600	470				
23	c39	600	510				

図 3.2: 図 2.2 のパターンをパターンクラスに分類

## 第4章 ハイブリッド法

ここでは、3章に引き続き分最適化アルゴリズムの高速化手法を提案する。最適化アルゴリズムのひとつである動的計画法は入力の部分問題を解いて記憶しておき、それらを用いてもとの問題を解く手法である。この手法は計算中に同一の部分問題を解く場合、記憶されている解を参照するだけでよいから、余分な計算をしない比較的高速なアルゴリズムである。しかし、入力部分問題をすべて記憶する必要があるため、実際には記憶容量の制限があり、入力サイズが大きい場合には適用できない。そこで、本研究では動的計画法の高速さを活用するために、分枝限定法に動的計画法を組み込んだハイブリッド法を提案する。

### 4.1 動的計画法

動的計画法を構成するためには、ピースの集合が与えられたときに、その解が参照できなければならない。そこで、ピースの集合の符号化方法を次のように定義する。

#### 定義 4.1 (ピースの集合の符号化)

整数を2進数で表現したときのビット幅を返す関数を  $\text{bit}()$ 、 $\text{shift}(j) = \sum_{i=j}^{|P|} \text{bit}(\text{num}(p_i))$  とする。このとき、符号化する関数を  $\text{key}(P) = \sum_{i=1}^{|P|-1} \text{num}(p_i) * 2^{\text{shift}(i+1)} + \text{num}(p_{|P|})$  と定める。

これにより、ピースの集合から一意に整数が得られる。この整数をキーとして解を記憶し、参照できるようにするために、ハッシュ表を次のように定義する。

#### 定義 4.2 (ハッシュ表)

ハッシュ表を部分解の集合  $H = \{\alpha_1, \alpha_2, \dots, \alpha_m\}$  と表し、 $|H| = m$  とする。このとき、あるピース集合  $P$  の解は、 $\alpha_{\text{key}(P)}$  である。

そして、動的計画法で解を計算するために、使用したビンの数に着目した部分解の集合を次のように定義する。

#### 定義 4.3 (部分解の集合)

$|\alpha| = m$  である部分解の集合を  $S_m = \{\alpha_1^m, \alpha_2^m, \dots, \alpha_n^m\}$  と表し、 $|S_m| = n$  とする。また、 $\text{set}(\alpha)$  を  $\alpha$  に含まれるピースの集合を得る関数と定める。

以上より、動的計画法はアルゴリズム 4.4 のように構成される。このように、ビン を 1 本使った解とビン を  $n-1$  本使った解からビン を  $n$  本使った部分解を計算していくときに、すでに計算した部分解を定数時間で参照することで計算効率を高めている。これは逆に、ハッシュ表に、入力  $P$  のすべての部分集合の最適解を格納しなければいけないということになる。したがってハッシュ表に必要なサイズ  $|H|$  は、各ピース  $p$  に対して  $\text{num}(p) + 1$  通りの選び方があるから  $|H| = \prod_{i=1}^{|P|} (\text{num}(p_i) + 1)$  となり、実際にはこの記憶容量の制限を受ける。

#### アルゴリズム 4.4 (動的計画法)

```
begin
  パターンの集合  $C$  から  $S_1$  を得て、 $S_1$  の要素をハッシュ表へ格納
  for  $n \leftarrow 2$  to ハッシュ表に解が格納されるまで
    for  $i \leftarrow 1$  to  $|S_1|$ 
      for  $j \leftarrow 1$  to  $|S_{n-1}|$ 
         $\alpha_m^n \leftarrow \alpha_i^1 \alpha_j^{n-1}$ 
        if  $\alpha_m^n$  が解または部分解
          then if ハッシュ表に  $\alpha_{\text{key}(\text{set}(\alpha_m^n))}$  が存在しない
            then  $\alpha_m^n$  を  $S_n$  に追加し、ハッシュ表へ格納
          else if  $\text{cost}(\alpha_m^n) < \text{cost}(\alpha_{\text{key}(\text{set}(\alpha_m^n))})$ 
            then  $\alpha_m^n$  を  $S_n$  に追加し、ハッシュ表へ格納
          end
        end
      end
    end
  end
end
```

## 4.2 ハイブリッド法

動的計画法には記憶容量の制限があり、入力サイズが大きい場合には適用できない。そこで、本研究では入力の部分集合を動的計画法で解き、残りを分枝限定法で解くハイブリッド法を提案する。

#### アルゴリズム 4.5 (ハイブリッド法)

1.  $P$  を互いに素である部分集合  $P_1, P_2, \dots, P_m$  に分割
2.  $P_1, P_2, \dots, P_n (n \leq m)$  を動的計画法で解き、それぞれの部分解集合をハッシュ表  $H_1, H_2, \dots, H_n$  へ格納
3. 探索木から  $\text{set}(c) \subset P_i (1 \leq i \leq n)$  を満たすノードを除く
4. 分枝限定法で  $H_1, H_2, \dots, H_n$  を利用しながら探索木をたどる

アルゴリズム 4.5 はハイブリッド法の流れである。ハイブリッド法では、探索木のあるノード  $c$  において、ピンに入っていないピースの集合  $P'$  が  $P' \subset P_i (1 \leq i \leq k)$  を満たすとき、 $H_i$  に記憶されている解を  $\text{key}(P')$  で参照し、ノード  $c$  の計算状況  $\gamma$  に加えることで、ノード  $c$  以下で得られるコストが最小の解を得ている。

図 4.1 は分枝限定法とハイブリッド法において計算した際のパターン数と探索ノードを比較した表である。ハイブリッド法は入力の部分集合の  $p_1 \cdots p_5$  と  $p_6 \cdots p_{10}$  を動的計画法で解いてそれぞれのすべての部分集合の解をハッシュ表に格納している。ハッシュ表のサイズはそれぞれ 384、720 である。また、 $p_1 \cdots p_5$  のみからなるパターンと、 $p_6 \cdots p_{10}$  のみからなるパターンを除いて探索木を構成している。分枝限定法との探索ノードを比較すると、ハイブリッド法により大幅に高速化されているのがわかる。なお、 $P$  を分割する方法によって、ハッシュ表を参照する回数やパターン数、探索ノード数は変動するため、適切な分割法を定める必要がある。

B	len(b)
b1	900
b2	600
b3	300

P	len(p)	num(p)		H
p1	550	2	H1	384
p2	510	3		
p3	470	1		
p4	410	3		
p5	380	3	H2	720
p6	320	4		
p7	290	2		
p8	240	3		
p9	170	2		
p10	130	3		

	C	探索ノード
ハイブリッド法	69	3012831
分枝限定法	165	1135691243

図 4.1: 分枝限定法とハイブリッド法の比較



## 第5章 高精度近似アルゴリズム

3、4章で最適化アルゴリズムを高速化する探索範囲縮小法とハイブリッド法を示した。しかし、実際には現実的な時間内で計算できない入力サイズが与えられる場合があるため、ここでは最適化が困難な場合に対する近似アルゴリズムについて述べる。本研究で提案する高精度近似アルゴリズムは、漸近的多項式時間近似スキームの考え方を応用させた手法である。

### 5.1 漸近的多項式時間近似スキーム

文献 [2] に、 $|B| = 1$ 、 $\text{len}(b_1) = 1$  のとき、 $0 < \epsilon \leq 1/2$  となる任意の  $\epsilon$  に対して近似保証  $(1 + 2\epsilon)\text{OPT} + 1$  を達成する以下の多項式時間アルゴリズム  $A_\epsilon$  が存在することが示されている。なお、 $\text{OPT} = \text{cost}(\alpha^*)$  である。

**アルゴリズム 5.1** ( $|B| = 1$  のときの漸近的多項式近似スキーム)

1. サイズ  $\epsilon$  未満のピースを除く
2. サイズを補正してピースのサイズを定数個にする
3. 最適パッキングを求める
4. このパッキングを最初のサイズのパッキングとする
5. サイズ  $\epsilon$  未満のピースを First-Fit で詰め込む

このアルゴリズムの特徴は、サイズ  $\epsilon$  未満のピースを除いた問題に対する最適パッキングを求める時間を多項式時間とみなしており、 $\epsilon$  を小さくして近似精度を 1 に近づけている点である。またサイズの補正と First-Fit によるピースの詰め込み方から近似保証  $(1 + 2\epsilon)\text{OPT} + 1$  を導いている。ただし、実際には  $\epsilon$  を 0 に近づけることは計算時間の面で困難であり、 $|B| \geq 1$  の場合は  $(1 + 2\epsilon)\text{OPT} + 1$  を保証していないため、このアルゴリズムを建築パーツ切り出し問題に適用しても実用的に優れた近似精度は期待できないと考えられる。

### 5.2 高精度近似アルゴリズム

本研究ではアルゴリズム 5.1 の考え方を応用し、 $|B| \geq 1$  の場合に対する高精度近似アルゴリズムを提案する。このアルゴリズムの流れは以下のようになる。まず、 $\epsilon$  未満のピースを除いて最適パッキングを求めたあと、 $\epsilon$  未満のピースに詰め込む際に、条件を満たすピースのみを詰め込んで部分的な暫定解を得る。次に、その暫定解から基準となる使用率を満たすパターンだけを解として確定し、これを除いた問題に対して再度計算を繰り返していく。以下にこのアルゴリズムを示す。

### アルゴリズム 5.2 (高精度近似アルゴリズム)

1. 最適化が可能なように  $\epsilon$  未満のピースを除く
2. 最適パッキングを求める
3. サイズ  $\epsilon$  未満のピースをアルゴリズム 5.3 で詰め込み暫定解を得る
4. 暫定解の中から使用率  $\beta$  以上のパターンを確定解とする
5. 確定解が得られれば 1 へ戻る
6. 確定解が得られなければ  $\beta$  を補正し 1 へ戻る

このアルゴリズムの 4 番目の処理の使用率とは、 $\sum_{i=1}^{|c|} \text{piece}(c)_i / \text{bin}(c)$  のことである。なお、確定解が得られないときは  $\beta$  を減少させていくため、このアルゴリズムは必ず停止する。次に、暫定解を得るアルゴリズムを以下に示す。

### アルゴリズム 5.3 (暫定解を得るアルゴリズム)

1. Optimal-Fit で詰め込む
2. 追加されたピースがあれば終了
3. Singular-Improve-Fit で詰め込む
4. 追加されたピースがあれば終了
5. Plural-Improve-Fit で詰め込む
6. 追加されたピースがあれば終了
7. First-Fit で詰め込む

Optimal-Fit、Singular-Improve-Fit、Plural-Improve-Fit はピースを詰め込む条件があるため、ピースを詰め込まない場合がある。一方、First-Fit はビンに詰め込む空きがあれば詰め込み、なければ新たなビンに詰め込む手法である。このように、詰め込み方に優先順位をつけて暫定解を得る。次に、Optimal-Fit、Singular-Improve-Fit、Plural-Improve-Fit の説明のために、以下を定義する。

#### 定義 5.4 (等間隔なビンの集合)

すべての  $i \in \{1, 2, \dots, |B| - 1\}$  で  $\text{len}(b_i) - \text{len}(b_{i+1})$  の値が等しいとき、 $B$  は等間隔であるといい、この間隔を  $\Delta$  で表す。

#### 定義 5.5 (パターンの余り)

パターン  $c$  の余りを  $\text{rest}(c) = \text{bin}(c) - \sum_{i=1}^{|c|} \text{piece}(c)_i$  と定める。

以上の定義から、Optimal-Fit、Singular-Improve-Fit、Plural-Improve-Fit をアルゴリズム 5.6、5.7、5.8 に示す。

#### アルゴリズム 5.6 (Optimal-Fit)

1.  $B$  が等間隔でなければ終了する
2. サイズ  $\epsilon$  未満のピースで  $\Delta < \text{len}(p)$  を満たすピース  $p$  と暫定解のパターン  $c$  を選ぶ
3.  $c$  に  $p$  を詰め込んで  $c'$  にする
4.  $\text{rest}(c') \leq \text{rest}(c)$  ならば  $c'$  を確定し 2 へ戻る
5.  $\text{rest}(c') > \text{rest}(c)$  ならば  $c'$  を取り消し 2 へ戻る

#### アルゴリズム 5.7 (Singular-Improve-Fit)

1. サイズ  $\epsilon$  未満のピース  $p$  と暫定解のパターン  $c$  を選ぶ
2.  $c$  に  $p$  を詰め込んで  $c'$  にする
3.  $\text{rest}(c') \leq \text{rest}(c)$  ならば  $c'$  を確定し 1 へ戻る
4.  $\text{rest}(c') > \text{rest}(c)$  ならば  $c'$  を取り消し 1 へ戻る

#### アルゴリズム 5.8 (Plural-Improve-Fit)

1. サイズ  $\epsilon$  未満のピース集合から構成されるパターン  $c_1$  と暫定解のパターン  $c_2$  を選ぶ
2.  $c_2$  に  $c_1$  のピースを詰め込んで  $c'$  にする
3.  $\text{rest}(c') \leq \text{rest}(c_2)$  ならば  $c'$  を確定し 1 へ戻る
4.  $\text{rest}(c') > \text{rest}(c_2)$  ならば  $c'$  を取り消し 1 へ戻る

Singular-Improve-Fit、Plural-Improve-Fit は暫定解のパターンの余りを改善するが、暫定解の最適性は保存しない。これに対し、Optimal-Fit には以下の性質がある。

#### 定理 5.9 (Optimal-Fit の最適性保存)

Optimal-Fit は暫定解の最適性を保存したままピースを詰め込むアルゴリズムである。

#### 証明

いま、 $B$  が等間隔で、 $P$  の部分集合に対する最適パッキング  $\alpha$  に  $\Delta < \text{len}(p)$  を満たす  $p$  を詰め込んで  $\alpha'$  を得たとする。このとき、 $\alpha'$  が最適パッキングでないという仮定から矛盾を導いて証明する。

$\alpha'$  は最適パッキングでないとすると、 $p$  を詰め込んだ解で  $\text{cost}(\alpha') - \text{cost}(\alpha'') \geq \Delta$  となる  $\alpha''$  が存在する。一方、 $\alpha''$  から  $p$  を除くと  $\Delta \leq \text{len}(p)$  であるから、 $\text{cost}(\alpha) > \text{cost}(\alpha^+)$  となる  $\alpha^+$  が得られる。

これは  $\alpha$  が最適パッキングであることに矛盾する。したがって、 $\alpha'$  が最適パッキングでないという仮定は誤りである。よって、Optimal-Fit は暫定解の最適性を保存したままピースを詰め込むアルゴリズムである。□

## 第6章 実験結果

ここでは、株式会社鈴工より提供された入力データ50件をもとに提案手法と従来手法の性能の比較を行う。入力サイズは図1.5、1.6に示したように、種類数で分類すると $|P| < 30$ のデータが全体の約75%、総数で分類すると $\sum_{i=1}^{|P|} \text{num}(p_i) < 100$ のデータが全体の約49%、 $100 \leq \sum_{i=1}^{|P|} \text{num}(p_i) < 200$ のデータが全体の22%であった。 $B$ は全データに対して、図1.3の木材のリストのデータを使用する。よって、 $|B| = 7$ で、 $\text{len}(b_1) = 2438$ 、 $\text{len}(b_2) = 3048$ 、 $\text{len}(b_3) = 3657$ 、 $\text{len}(b_4) = 4267$ 、 $\text{len}(b_5) = 4876$ 、 $\text{len}(b_6) = 5486$ 、 $\text{len}(b_7) = 6096$ である。なお、ここでは $B$ を等間隔であるとし、 $\Delta \simeq 610$ とする。また、プログラムはJavaで実装し、インテル Core2Duo プロセッサ P8400(2.26GHz)のCPUを搭載した市販のパーソナルコンピュータで実行した。

図6.1は入力データ50件に対するアルゴリズム別の実行結果を示した表である。ここで $\text{sum}(P)$ はピースの総数、高速最適化アルゴリズムは探索範囲縮小法とハイブリッド法を併用した手法としている。また、分枝限定法と高速最適化アルゴリズムの列の○は1時間以内に最適解を計算できたことを意味しており、高精度近似アルゴリズム、グリーディー法、FFDLRまたはFFDLSの列の実数は定義2.5で定めた近似度を表している。

### 6.1 高速最適化アルゴリズムの評価

入力データ50件に対して1時間以内に最適解を計算できた件数は、分枝限定法が8件であるのに対し、高速最適化アルゴリズムの場合は21例であり最適化可能な入力サイズの許容範囲を広げる結果が得られた。

### 6.2 高精度近似アルゴリズムの評価

入力データ50件に対して、近似度平均はグリーディー法が1.011、FFDLRまたはFFDLSは1.033であるのに対し、高精度近似アルゴリズムは1.005であった。また、グリーディー法では精度にばらつきがみられたが、高精度近似アルゴリズムでは比較的安定した近似精度を達成している。たとえば入力番号25ではグリーディー法の近似精度が1.032となっているが、高精度近似アルゴリズムの近似度は1.007である。これはグリーディー法が短いピースを最初に使用してしまい、後半に余りの多いパターンが出てきたことが原因と考えられる。一方、高精度近似アルゴリズムでは暫定解に $\epsilon$ 未満のピースを詰め込むため、長いピースと短いピースを比較的バランスよく選択していくため、このような結果が得られたと考えられる。なお、高精度近似アルゴリズムの計算時間はいずれも数十分程度であった。

### 6.3 まとめ

以上の結果から、本研究では従来よりも、最適化可能な入力サイズの許容範囲を広げ、安定した高精度な近似精度を達成する実用的なアルゴリズムを提案することができた。

入力番号	P	sum(P)	分枝限定法	高速最適化アルゴリズム	高精度近似アルゴリズム	グリーディー法	FFDLRまたはFFDLS
1	11	83		○	1	1.015	1.035
2	5	33	○	○	1	1	1
3	16	197			1.034	1.042	1.044
4	6	161		○	1	1	1
5	5	73	○	○	1	1.002	1.002
6	7	57		○	1	1.012	1.024
7	12	91			1.016	1.026	1.045
8	5	72	○	○	1	1	1
9	10	94			1.01	1.01	1.034
10	9	76		○	1	1	1.005
11	7	65	○	○	1	1	1.002
12	7	78		○	1	1	1.002
13	8	125		○	1	1.001	1.006
14	7	79	○	○	1	1.001	1.006
15	6	91		○	1	1	1
16	5	86		○	1	1	1
17	9	64		○	1	1	1.02
18	10	63		○	1	1	1.027
19	8	66		○	1	1.004	1.007
20	8	68		○	1	1.007	1.018
21	5	56	○	○	1	1	1
22	3	58	○	○	1	1	1.033
23	8	66		○	1	1.004	1
24	5	72	○	○	1	1	1
25	16	118			1.007	1.032	1.062
26	19	65			1.028	1.034	1.06
27	53	228			1.005	1.025	1.046
28	20	72			1.009	1.04	1.074
29	90	330			1.003	1.013	1.049
30	67	434			1.001	1.011	1.037
31	51	606			1.003	1.002	1.051
32	38	259			1.006	1.009	1.032
33	16	36			1.013	1.025	1.037
34	14	408			1.003	1.003	1.032
35	37	650			1.005	1.013	1.044
36	50	458			1.015	1.017	1.065
37	24	116			1.002	1.019	1.074
38	24	177			1.001	1	1.061
39	82	332			1.007	1.015	1.058
40	36	107			1.007	1.013	1.088
41	25	314			1.012	1.049	1.058
42	43	403			1.017	1.019	1.067
43	25	95			1.008	1.01	1.039
44	14	114			1.003	1.003	1.059
45	49	461			1.01	1.015	1.028
46	21	226			1.007	1.007	1.042
47	29	220			1.021	1.032	1.054
48	8	142			1.016	1.016	1.044
49	13	152			1.009	1.014	1.06
50	91	263			1.003	1.02	1.058

図 6.1: 各アルゴリズムの実行結果

## おわりに

本研究で建築パーツ切り出し問題のための実用的なアルゴリズムとして高速最適化アルゴリズムと、高精度近似アルゴリズムを提案することができた。今後の改善案として、まず枝刈りの工夫が挙げられる。現在は上界値と下界値を用いた手法のみであるが、あるノードにおける計算状況 $\gamma$ のコストを改善することが出来る場合はそのノードの探索は省略可能である。しかし、ノードをたどる度にこの判定をすると判定時間がかかってしまうため、工夫しなければいけない。また、パターンのインデックスのつけ方により、パターンクラスの数が変わることがあためパターンクラスを最大化するインデックスのつけ方について考察するべきである。さらに、ハイブリッド法において、ピースの分割法はパターン数、ハッシュ表の利用率、探索ノード数に影響するため、適切な分割法についての議論も重要である。なお、ハッシュ表を効率よく活用するために分枝限定法の途中で得られた解をハッシュ表に記憶させるという高速化手法も考えられ、これについては実装・実験による評価が必要と考えられる。高精度近似アルゴリズムについてはアルゴリズム 5.2 における基準値 $\beta$ の定め方、暫定解にピースを詰め込む方法についての妥当性を示すためにさらに多くの入力データに対して実験を行い、解析・改善をする必要があり、改善策として出力した解を局所最適化して近似精度を向上させる手法が考えられる。

## 参考文献

- [1] D. K. Friesen and M. A. Langston "Variable Sized Bin Packing" SIAM J. COMPUT Vol.15, No 1, February (1986), pp.222-229
- [2] Vijay V. Vazirani, "Approximation Algorithms", Springer, 2001, pp.74-78
- [3] 平田富夫, "アルゴリズムとデータ構造", 森北出版, 1990
- [4] L. Epstein and A. Levin. An algorithm for generalized cost variable-sized bin packing. SIAM J. Comput, 38(1):pp.411-428, 2008
- [5] J. Kang and S. Park. Algorithms for the variable sized bin packing problem. European Journal of Operational Research, 147, pp.365-372, 2003
- [6] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In Proc. 23rd Annual IEEE SFOCS, pp.312-320, 1982
- [7] F. D. Murgolo. An efficient approximation scheme for variable-sized bin packing. SIAM J. Comput, 16(1):pp.149-161, 1987

## 謝辞

本論文の作成にあたり、研究指導をしてくださった大山口道夫教授、共同研究者として協力してくださった三橋一郎氏、入力データを提供してくださった株式会社鈴工様をはじめ、日頃からお世話になっている、山田俊行講師、落合美子事務員、所属研究室の院生、学部生の皆様に深く感謝の意を表します。