修 士 論 文

最適木取りアルゴリズムに関する研究

平成 23 年度 修了
三重大学大学院　工学研究科
博士課程前期課程　情報工学専攻
計算機ソフトウェア研究室

柳島弘章

概 要

住宅等を建築する際に使用する数十から数百本の柱（建築パーツ）は前もって材木から一括して切り出され，このときに木材の不用な余りが発生する．この余りを最小にする柱の組み合わせを求めるアルゴリズムは建築パーツ切り出し問題と呼ばれ，高速で最適解を求める手法が実際の現場で求められている．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り，NP 完全問題として知られている [FL86]．従って，この問題は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難しくなる．しかし，建築パーツ切り出し問題のような実際の問題への適用では現実的な時間で計算できる場合が多くあり，最適化アルゴリズムを高速化して，より多くの入力例に対して最適解が得られるように改善することは，有益である．

先行研究として野呂の研究 [野呂 10] がある．野呂はパターンクラスの概念の導入やハッシュ表を利用した分枝限定法を用いた高速最適化アルゴリズム，及び，近似アルゴリズムとして，可変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与えた．ここで，パターンとは，材木とそこから切り出される柱のリストのペアであり，それぞれのアルゴリズムはパターンのリスト（即ち，柱の切り出し方）を出力する．

本研究ではまず野呂の高速最適化アルゴリズムにおいて提案されたパターンクラスの概念をより一般化することによりパターンクラス数を減らすことに成功し，分枝限定法における探索節点数が削減できることを実証した．次に，分枝限定法で求めた部分最適解をハッシュ表に保存して再利用する動的ハッシュ法を適用した．また，部分最適解が得られなかった場合にはハッシュ表に枝刈り可能な情報を格納し，以後，その値を利用して枝刈りを行うようにアルゴリズムを改善した．さらに，入力から生成されるパターンに対して分割可能なパターンという概念を導入して，パターンの総数及びその探索範囲を大幅に削減した．これらの新しい手法を導入することにより野呂らの高速最適化アルゴリズムを高速化すると共に，現実的な時間で最適解を計算可能な入力のサイズを広げた新しいアルゴリズムを提案した．

本研究では提案した手法の有効性を確認するために，本アルゴリズムを実装して実際の現場で使われているデータ（49 例）を用いて先行研究のアルゴリズムと計算時間と探索接点数を比較する実験を行った． 49 例中 13 例に改善が見られた．そのなかで表.1 で示す 3 例については大幅な改善を確認することができ，本手法の有効性を実証した．

表 1: 従来手法との比較

			提案手法		従来手法	
入力例	$\|P\|$	sum(P)	探索節点数	探索時間	探索節点数	探索時間
7	12	91	343628	4.9	394719936	675.0
9	10	91	491704	7.3	62516144	105.9
26	19	65	35600	1.0	127946915	218.4

目 次

第1章　建築パーツ切り出し問題

木造住宅等を建築するためには数十から数百本の柱（建築パーツ）が必要である．建築業者により設計された設計図を元に柱の長さと必要な本数が決定される．建築業者は必要な柱の長さと本数を木材加工業者に伝える．木材加工業者は原木から一定の長さに切りそろえた材木として保存している．保存されている材木の中から適当に材木を選び木材加工用の工作機械を用いて必要な柱を切り出す．このとき, ストックされている材木の中からどのように切り出すかは非常に多くの組み合わせがあり, 切り出し方は職人の判断にまかされているため多くの無駄が発生している．

300mm の材木のストックがあり 100mm の柱が 3 本必要な時の切り出し方法としては, 例えば表.1.1 や表.1.2 のようなものが考えられる．表.1.1 は 300mm の材木が 2 つ, 合計で 600mm の材木が必要である．それに対して表.1.2 では 300mm の材木 1 本で必要な柱を切り出せている．このような単純な例であれば無駄な材木をできるだけ使用しない, 最適な切り出し方法は容易に導き出せるが, 数が多くなると最適な切り出し方法の算出は困難である．

表 1.1: 切り出し方の例

材木	柱	柱
300	100mm	100mm
300	100mm	

表 1.2: 切り出し方の例 2

材木	柱	柱	柱
300	100mm	100mm	100mm

ここで柱を切り出すための材木は十分な本数が用意されているものとする．長さの長い材木は樹齢の長い木を伐採するため一般的には高価となるため材木のコストは必ずしも長さには比例しないが, ここではコストは長さに比例するものとする．先行研究の実験結果から材木のコストは長さに比例すると考えて良いことが分かっている．このとき, 柱を切り出したときにかかるコストは使用した材木の長さの総和となる．

本研究では、「材木の長さのリスト、建築パーツの長さと必要数のリストが与えられたとき、使用した材木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．

本研究は木材加工機メーカである株式会社鈴工より依頼があり行われている研究である．本研究の有効性を実証するために使用する入力データは同社より提供された実際の現場で使われているものを用いている．表.1.3, 表.1.4 にそれぞれ実際に提供された材木のストックと木造住宅建築一軒分に必要な柱のデータの一例をあげる．

表 1.3: 材木のリスト

長さ
6096mm
5486mm
4876mm
4267mm
3657mm
3048mm
2438mm

表 1.4: 柱の例

長さ	必要な本数
4789mm	10 本
3841mm	4 本
3651mm	11 本
3607mm	9 本
3196mm	11 本
2741mm	11 本
2242mm	6 本
1844mm	3 本
1780mm	3 本
1736mm	16 本
915mm	4 本
870mm	3 本

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り，NP 完全問題として知られている．[FL86] したがって，この問題は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難しくなる．しかし，建築パーツ切り出し問題のような実際の問題の適用では現実的な時間で計算できる場合が多くあり，最適化アルゴリズムを高速化して，より多くの入力例に対して最適解が得られるように改善することは，有益である．

表 1.4 の種類と必要な総本数はそれぞれ,12 種類と 91 本である．この必要な柱のリストと表 1.3 を入力としてこの程度の本数であれば最適解をみつけることが容易であるようの思うが, 実際は野呂の高速最適化アルゴリズムで 675 秒（約 11 分）かかってしまう．この例を最適化アルゴリズムで計算した結果を表.1.5 に示す．また, 野呂が提案した近似アルゴリズム, 高精度近似アルゴリズムを用いても近似的な解を計算した場合でも 531 秒ほどかかってしまう．高速最適化アルゴリズムを実行したコンピュータは Core2Quad Q9450(2.66GHz), 高精度近似アルゴリズムを実行したコンピュータは Core2Duo P8400(2.26GHz) を搭載するコンピュータであり, 一概には比較できないが時間がかかると言う事に違いはない.

表 1.5: 解の例

材木	柱	柱	柱	柱	不要な余り
4876mm	4789mm				87mm
4876mm	4789mm				87mm
4876mm	4789mm				87mm
4876mm	4789mm				87mm
4876mm	4789mm				87mm
4876mm	4789mm				87mm
4876mm	4789mm				87mm
4876mm	4789mm				87mm
4876mm	4789mm				87mm
4876mm	4789mm				87mm
4876mm	3841mm	915mm			120mm
4876mm	3841mm	915mm			120mm
4876mm	3841mm	915mm			120mm
4876mm	3841mm	915mm			120mm
3657mm	3651mm				6mm
3657mm	3651mm				6mm
3657mm	3651mm				6mm
3657mm	3651mm				6mm
3657mm	3651mm				6mm
3657mm	3651mm				6mm
3657mm	3651mm				6mm
3657mm	3651mm				6mm
3657mm	3651mm				6mm
3657mm	3651mm				6mm
3657mm	3651mm				6mm
5486mm	3607mm	1844mm			35mm
3657mm	3607mm				50mm
3657mm	3607mm				50mm
3657mm	3607mm				50mm
3657mm	3607mm				50mm
3657mm	3607mm				50mm
3657mm	3607mm				50mm
3657mm	3607mm				50mm
3657mm	3607mm				50mm
6096mm	3196mm	2741mm			159mm
6096mm	3196mm	2741mm			159mm
6096mm	3196mm	2741mm			159mm
6096mm	3196mm	2741mm			159mm
6096mm	3196mm	2741mm			159mm
5486mm	3196mm	2242mm			48mm
5486mm	3196mm	2242mm			48mm
5486mm	3196mm	2242mm			48mm
5486mm	3196mm	2242mm			48mm
5486mm	3196mm	2242mm			48mm
5486mm	3196mm	2242mm			48mm
5486mm	2741mm	2741mm			4mm
5486mm	2741mm	2741mm			4mm
5486mm	2741mm	2741mm			4mm
3657mm	1844mm	1780mm			33mm
3657mm	1844mm	1780mm			33mm
3657mm	1780mm	1736mm			141mm
6096mm	1736mm	1736mm	1736mm	870mm	18mm
6096mm	1736mm	1736mm	1736mm	870mm	18mm
6096mm	1736mm	1736mm	1736mm	870mm	18mm
3657mm	1736mm	1736mm			185mm
3657mm	1736mm	1736mm			185mm
3657mm	1736mm	1736mm			185mm

第2章　可変サイズビンパッキング問題

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. 可変サイズビンパッキング問題は無限の個数が用意されており長さが異なる有限種類のビンに有限種類, 有限個のピースを詰めて消費するビンの長さの総和が最小となる組み合わせを求める問題であり, 組み合わせ最適化問題の一種である.

建築パーツ切り出し問題における材木と柱（建築パーツ）は，それぞれ可変サイズビンパッキング問題におけるビンとピースに対応している．これ以後, 可変サイズビンパッキング問題の定義に従って記述する. 本章では, その諸定義について述べる。

定義 2.1 (ビンとピース) ビンの種類のリストを $B = [b_1, b_2, ..., b_{m_0}]$，ピースの種類のリストを $P = [p_1, p_2, ..., p_{n_0}]$，ビンとピースの長さを表す関数として $len()$，ピースの本数を表す関数として $num()$ を定義する．ビンはそれぞれ無限個のストックがあるものとする．**ピース長の総和**は $\Sigma_{i=1}^{m_0} len(p_i) * num(p_i)$ で与えられるものとする．ピースの本数の総和を $Sum(P) = \Sigma_{i=1}^{|P|} num(p_i)$ と定義する.

定義 2.2 (パターン) ビンにピースを詰める組み合わせをパターンと呼び $c = (b, q_1, q_2, ..., q_n)$ $(\in B \times P^*)$ と表記する．$|c| = n$ とする．ただし，b はパターン c に含まれるピースの長さの総和以上の最小のビンであり, ピースは長さの降順に並んでいる物とする．パターンに含まれる i 番目のピースの本数を $count(c,\ i)$ とする.$bin(c) = len(b)$ とする．c のピース長のベクトルを $lvec(c) = (len(q_1),\ len(q_2),\ ..., len(q_n))$ とする．c の余りを $rest(c) = b - \Sigma_{i=1}^{|c|} len(q_i)$ とする．パターン c に含まれるビン b やピース $q_1,\ ...,\ q_n$ は次の方法で取得する.

$$c(i) = \begin{cases} b & (i = 0) \\ q_i & (1 \leq i \leq n) \end{cases}$$

パターン c に含まれるピースの集合を返す関数を $piece(c) = \{c(i) \mid 1 \leq i \leq |C|\}$ とする.

定義 2.3 (パターンの大小関係) パターン $c,\ c'$ について各 i $(1 \leq i \leq min(|c|,\ |c'|))$ について $c(i) \leq c'(i)$ であるとき $c \leq c'$ であると定義する.

定義 2.4 (パターンの列と解) パターンの列を $\alpha = d_1, ..., d_m$ と表し，辞書式順序の降順に並んでいるものとする．解のコストを $cost(\alpha) = \Sigma_{j=1}^{m} bin(d_j)$ とする．パターン列 α に i $(1 \leq i \leq |P|)$ 番めピース p_i が含まれている本数を返す関数を $num(\alpha,\ i)\ = \Sigma_{j=1}^{m} count(d_j, i)$ とする．全ての i $(1 \leq i \leq |P|)$ に対して，$num(\alpha,\ i) \leq\ num(p_i)$ が成り立つとき α を**部分解**であるといい，$num(\alpha,\ i) =\ num(p_i)$ を満たす α は**解**であるという.

定義 2.5 (近似度) 解 α の近似度を $cost(\alpha)/cost(\alpha^*)$ と定める.α^* は最適解である．ただし，α^* を得るのが困難な場合は $cost(\alpha)/$ **ピース長の総和**を用いる.

第3章　先行研究

本研究の先行研究として野呂 [野呂 10] の研究がある．野良は建築パーツ切り出し問題に対する最適化アルゴリズムと近似アルゴリズムを提案した．最適化アルゴリズムとして分枝限定法にパターンクラスという概念を導入することにより高速化した探索範囲縮小法, 動的計画法を併用したハイブリッド法を与えた．現実的な時間では計算できない入力に対して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した．野呂らの手法では分枝限定法をさらに高速化するために次で定義されるパターンクラスの概念を導入した探索範囲縮小法を提案した．

3.1　分枝限定法

全ての組み合わせを列挙して探索する方法として探索の対象を木構造で表現し深さ優先（もしくは幅優先）探索で探索する手法がある．このとき，初期解を与えて解の上界値として設定し，解の暫定解が上界値を越えることが探索の過程でわかったとき, それ以降の探索を打ちきるという手法は分枝限定法としてしられている．

定義 3.1 (最長のピースが等しいパターンの集合) 入力からパターンの集合 C が生成されたとき, パターンにおける最も長いピースが p_i である集合を $C_i = \{c \in C \mid c(1) = p_i\}$ とする．

節点がパターンの集合 $C = \{c_1, c_2, ..., c_{n_0}\}$ の要素である探索木が構成される．全ての i $(1 \leq i \leq n_0 - 1)$ について,$c_i \geq c_{i+1}$ が成立しているものとする．ある探索節点を探索したとき子供として選ぶ節点はパターンの集合 C の要素であり, かつ未使用のピースのうちで長さが最も長いピースが p_i であるときに C_i の要素に限る．各節点 c の**計算状況**を根からその節点までのパス上のパターンの列と定義する．また, **計算状況**が**部分解**であるときのみその子供の節点を探索する．

定義 3.2 (下界値) ある節点 c において, 節点 c の計算状況 γ としたとき, 節点 c における下界値 L を次のように定める．すべての $i \in \{1, ..., |P|\}$ について $L = cost(\gamma) + \Sigma_{i=1}^{|P|} len(p_i) * (num(p_i) - num(\gamma,\ i))$.

定義 3.3 (上界値) 得られた解の中で, コストが最小の解を α とするとき, そのコストを上界値 $U = cost(\alpha)$ と定める．

定義 3.4 (枝刈り) ある節点 c における下界値 L と上界値 U に $L \leq U$ が成り立つとき、節点 c の探索は省略可能である。

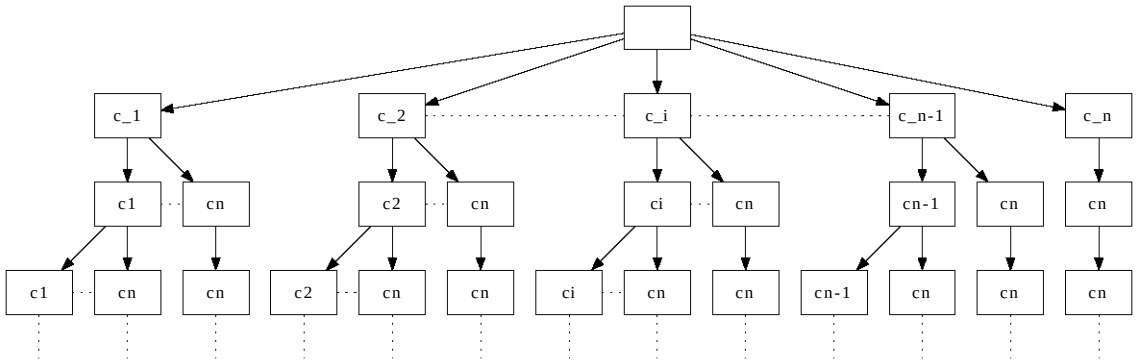

図 3.1: 探索木

3.2 探索範囲縮小法

探索範囲縮小法は分枝限定法にパターン間の包含関係とパターンクラスの概念を導入して高速化したアルゴリズムである. パターンクラスの概念を導入することにより分枝限定法における探索において兄弟の節点の探索を大幅に省略できることを明かにした. 野呂は以下で定義される包含関係を提案した.

定義 3.5 (パターン間の包含関係) 2つのパターン $d = (b, q_1, ..., q_k)$, $d' = (b', q'_1, ..., q'_l)$ について $len(b) = len(b')$ かつ $k \geq l$ であり, すべての $i \in \{1, ..., l\}$ について $len(q_i) \geq len(q'_i)$ であるとき d は d' を包含する $(c \sqsupseteq c')$ と定義する.

パターン c が**探索可能**であるときにパターン c が包含可能なパターン $c'(\sqsubseteq c)$ の探索を省略可能である. この性質からパターン c が探索可能であれば探索を省略可能なパターンの集合であるパターンクラスを野呂は次のように定義した.

定義 3.6 (パターンクラス) あるパターン c が包含可能なパターンの集合を c のパターンクラスと呼び, $\delta(c) = \{c' \mid c \sqsupseteq c' \text{ かつ } c \neq c'\}$ と表す.

パターン c が探索可能であれば, 探索木の兄弟に存在する節点のうちパターンクラス $\delta(c)$ の要素となるパターンを持つ節点の探索を省略でき, このことにより分枝限定法における探索木の幅 (探索範囲) を大幅に縮小に成功している.

第4章　提案手法

本研究ではまず野呂の高速最適化アルゴリズムにおいて提案されたパターンクラスの概念をより一般化することによりパターンクラス数を減らすことに成功し，分枝限定法における探索節点数が削減できることを実証した．次に，分枝限定法で求めた部分最適解をハッシュ表に保存して再利用する動的ハッシュ法を適用した．また，部分最適解が得られなかった場合にはハッシュ表に枝刈り可能な情報を格納し，以後，その値を利用して枝刈りを行うようにアルゴリズムを改善した．さらに，入力から生成されるパターンに対して分割可能なパターンという概念を導入して，パターンの総数及びその探索範囲を大幅に削減した．これらの新しい手法を導入することにより野呂らの高速最適化アルゴリズムを高速化すると共に，現実的な時間で最適解を計算可能な入力のサイズを広げた新しいアルゴリズムを提案した．

4.1　パターンクラスの一般化

本研究では野呂の提案した探索範囲縮小法におけるパターン間の包含関係とパターンクラスの概念を一般化した新たなパターンクラスを与える．また, 野呂の手法では無駄の多かったパターンクラスの実装ついて無閉路有向グラフ（以下 DAG と略す）を用いた手法でパターンクラスの探索範囲削減効果を最大限に活用する．このことにより１つのパターンクラスにおいて探索を省略可能なパターンの数を増加させ, パターンクラスの数を削減し, 分枝限定法の探索における兄弟の節点の探索を削減する．

定義 4.1 (パターンの間の包含関係) ２つのパターン $c = (b, q_1, ..., q_k)$，$c' = (b', q'_1, ..., q'_l)$ について，$len(b) = len(b')$ であり，ある関数 $f : \{1, ..., l\} \mapsto \{1, ..., k\}$ が存在して各 $i(1 \leq i \leq k)$ について $\Sigma_{j \in S_i}\ len(q'_j) \leq len(q_i)$ *(*ただし，$S_i = \{j \mid f(j) = i\}$ ）を満たすとき $c \sqsubseteq' c'$ である．

定義 4.2 (パターンクラス) パターン c のパターンクラスを $\beta(c) = \{c' \mid c \sqsupseteq' c'$ かつ $c \neq c'\}$

野呂らのパターンクラス δ との間には $\delta(c) \subset \beta(c)$ の関係がなりたち，野呂らのパターンクラスを真に包含しているため，探索範囲をより縮小していることがいえる．

証明 4.1 (一般化した包含関係により探索を省略しても最適解が得られる) あるパターン列 $\alpha^* = d_1, ..., d_n$ が存在して α^* は最適解であるとする.
ここで,α^* のあるパターン d_i を一般化した包含関係で包含可能な p ($\sqsupseteq' d_i$) が存在するとする. p が $piece(p) \subset \cup_{k=i}^{n} piece(d_k)$ を満たすならば, あるパターン $p_1, ..., p_n$ が存在して, α^* 中の d_i を p に交換した新たな解 $\beta^* = d_1, ..., d_{i-1}, p, p_1, ..., p_m$ が得られる.

1. d_i の子孫の節点にパターン c に含まれるピースの集合を d_i に追加して c', p を得る場合を考える. $p \sqsupseteq' d_i$ であることから $bin(p) = bin(d_i)$ が成立する. α^* が最適解であるという仮定から,c からピースを取り出してもビンの長さわ変わらず $bin(c') = bin(c)$ となる.

2. d_i の子孫の節点 c のピースと d_i のピースを交換して c', p を得る場合を考える. $p = (b, q_1, ..., q_k), d_i = (b, q'_1, ..., q'_k)$ であるとすると, $p \sqsupseteq' d_i$ であることから q_i に対応する d_i 中のピースの集合 D は $D(q_i) = \{q'_j \mid j \in S_i\}$ で与えられる. c の交換するピースを s, d_i の交換するピースの集合を $D(s)$ とすると, $p \sqsupseteq' d_i$ であるから $len(s) \geq \Sigma_{t \in D(s))}\, len(t)$ である. よって,c のピース s を長さの総和が $len(s)$ 以下となるピースの集合 $D(s)$ と交換することと,α^* が最適解であることから $bin(c') = bin(c)$ である. また, 同様に $bin(p) = bin(d_i)$ が成立する.

以上より,α^* 中の p ($\sqsupseteq' d_i$) を d_i で置き換えても β^* の最適性は保存されることが言える. α^* の節点 d_i を p に置き換えた β^* でも最適解が得られている. よって, 一般化した包含関係により探索を省略しても最適解が得られることが言えた.

4.1.1 一般化した包含関係を計算するアルゴリズム

あるパターン c, c' が $bin(c) = bin(c')$ であるとき, 一般化した包含関係 $c \sqsubseteq' c'$ が成り立つかと言う問題は $piece(c)$ をビン,$piece(c')$ をピースと考えたときの固定サイズビンパッキング問題において解が存在するかという問題と等価である. 包含関係の計算には近似アルゴリズムの FirstFit 法を用いて計算し適当な解が見つかれば包含するものとしている.

4.2 DAG によるパターンクラスの表現

野呂らの手法ではパターンクラスを列で表現していた. この方法は単純であり実装が平易な半面に無駄があり本来探索を省略されるべき節点が省略されない事がある. 表.4.1 に野呂の手法で生成されるパターンクラスの 1 例を示す. 野呂が定義した包含関係によるとパターン c_1 に対するパターンクラス $\delta(c_1)$ は $\delta(c_1) = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7\}$ となるが実際は次の表のようにパターンクラスが細かく分断されてしまっていて効率的に機能しているとは言い難い.

表 4.1: 野呂の手法におけるパターンクラス

パターンクラス	パターン	ビン	ピース 1	ピース 2	ピース 3	ピース 4
$\delta(c_1)$	c_1	900mm	320mm	320mm	130mm	130mm
$\delta(c_2)$	c_2	900mm	320mm	130mm	130mm	130mm
$\delta(c_3)$	c_3	900mm	320mm	240mm	130mm	130mm
$\delta(c_4)$	c_4	900mm	320mm	240mm	130mm	
$\delta(c_5)$	c_5	900mm	320mm	130mm		
$\delta(c_5)$	c_6	900mm	320mm			
$\delta(c_7)$	c_7	900mm	320mm	320mm	240mm	
$\delta(c_7)$	c_8	900mm	320mm	320mm		

本研究ではパターンクラスを DAG の形で表現することによりパターンクラスの効能をより大きく引き出している. 生成される DAG における節点はパターンであり，辺は一般化したパターン間の包含関係である. 一般化したパターンクラスでは c_1 に対するパターンクラス $\beta(c_1)$ は $\beta(c_1) = \{c_i \mid 1 \leq i \leq 8\}$ となる. 図.4.1 に提案手法において一般化したパターンクラスの定義から生成される DAG を示す. パターン c_1 が探索可能であれば探索を省略できる $\beta(c_1)$ の要素となる全てのパターンは DAG の c_1 の子孫の節点を探索することで得られる. このように DAG を構成して辿ることで, 探索を省略できる節点を先に探索してしまうことを避けている. 例えば $\beta(c_1) \subset c_3$ であるので,c_1 が探索可能であることが分かれば c_3 の探索を省略できるが, 逆の順序で探索すると省略できない.

実際に探索中に DAG を構成して辿っていたのでは時間がかかるため, 探索木においてある節点の子節点を探索するとき, 子節点から DAG を生成してトポロジカルソートした順序で探索する. 例えば, ある節点の子節点が表.4.1 で示したパターンであるとき, これらのパターンから図.4.1 のような DAG を生成しトポロジカルソートした順序で探索する. この場合の探索の順序は c_1, c_3, c_7, c_2, c_4, c_8, c_5, c_6 となる. この順序の逆に探索を行うと探索の省略が一切できなくなる.DAG を用いると, そのような自体を避けることができる.

4.3 ビンのサイズが違っても良い包含関係

先に提案したパターン間の包含関係は条件付けがなく一般的な場合に言える概念である. ここでは条件は厳しいものの特定の状況下においてパターンのビンの長さが違っても成立する包含関係を定義した. この包含関係が成り立つためには入力のビンが**等間隔**であることが前提である.

定義 4.3 (等間隔) すべての $i(1 \leq i \leq |B| - 1)$ で $len(b_i) - len(b_{i+1})$ が等しいとき,B は**等間隔である**と言い, この間隔を Δ と表現する.

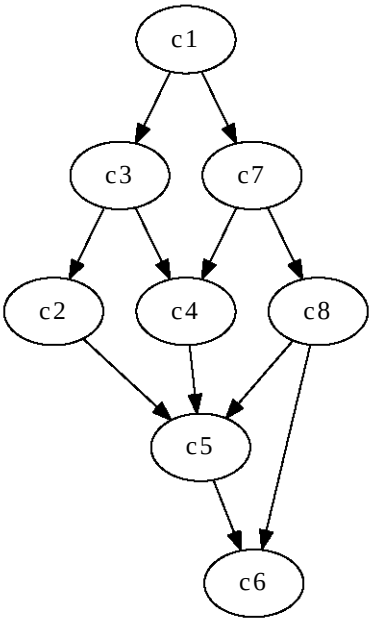

図 4.1: 一般化したパターンクラスの定義から生成される DAG

入力のビンが表.1.3 で与えられるビンのリストであり $\Delta \simeq 609mm$ で**等間隔**であるとき，最適解に含まれるビンの中で長さが 2438mm もしくは 3048mm のビンは高々 1 個であることがいえる．最適解にパターンの長さが 2438mm，または 3048mm のものが 2 つ以上含まれていたとしても，これら 2 つのビンを組み合わせることで 4876mm，5486mm，6096mm のビンにまとめた最適解が存在するためである．このことから探索木を，長さが 2438mm のビンを 1 つ選びそれ以降は長さが 3048mm 以下のビンを選ばない場合，同様に長さが 3048mm のビンを 1 つ選んだ場合，長さが 2438mm と 3048mm をまったく含まないものに分割することができる．このように探索木を分割したときに以下に定義した包含関係が成り立つ．

定義 4.4 (ビンサイズが違っても成立する包含関係) ビンの長さが**等間隔**である．探索が分割されているとき，2 つのパターン $c = (b, q_1, ..., q_l)$，$c' = (b', q'_1, ..., q'_k)$ について $len(b) - len(b') = \Delta$ かつ $k \leq l$ であり，全ての i $(1 \leq i \leq k)$ について $len(q_i) = len(q'_i)$ であるとき c は c' を包含する $(c \sqsupseteq'' c')$ と定義する．

4.4 分割可能なパターンの概念

これまでは, 分限定法の探索木において探索の過程で兄弟の節点の探索を省略するといった枝刈りの手法を提案してきた. ここでは**分割可能**なパターンという概念を導入し, 探索木を構成する前の入力であるビンとピースから生成されるパターンを探索木を構成する前に削減する手法を提案する. 最適解を得るために要する時間はパターンの数に大きく依存するため, パターンの数を減らすことで探索時間を大幅に縮小できる可能性がある. この手法ではパターンの数を増加させる要因となっている長さが小さいピースを多数含むようなパターンを削減できる. この手法を適用するには入力のビンの集合が**等間隔**である必要がある. ここでは実際に業者から提供されたビンのデータを用いる.

4.4.1 分割可能なパターン

あるパターン c に含まれるピースを分割して, 新たなビンに詰めてパターン c_1 と c_2 とする. このとき, パターン c を探索木から取り除き, 分枝限定法における探索木の節点としてはパターン c の代わりにパターン c_1 を考える. パターン c_1 からパターン c が復元可能なときには復元する. パターン c の代わりにパターン c_1 を考えるので, パターンが減っていないように思えるかもしれないがパターン c_1 は元より入力から生成されるパターンの集合に含まれているため, パターン c が削除されるのみであり, 新たに c_1 という新しいパターンが追加されるわけではない. このようにしても最適解が得られるとき, パターン c を**分割可能**であるといい次のように定義する.

定義 4.5 (分割可能) ビンの長さがそれぞれ$len(b_1) = 6096mm, len(b_2) = 5486mm, len(b_3) = 4876mm, len(b_4) = 4267mm, len(b_5) = 3657mm, len(b_6) = 3048mm, len(b_7) = 2438mm$ であるときビンの列 $B = [b_1, b_2, ..., b_7]$ を考える. これらのビンは $\Delta \simeq 609mm$ で**等間隔である**.
このとき, あるパターン $c = (b, l_1, ..., l_n)$ が存在する.
パターン c のピースを取り出して作成したパターンを $c_1 = (b_1, l'_1, ..., l'_m)$ とパターン $c_2 = (b_2, l''_1, ..., l''_r)$ とする. ここで $r + m = n$ である. このとき $Max(l'_i) > Max(l''_i)$ かつ, $rest(r_1) \leq rest(r)$ かつ, $\Sigma\ len(l''_i) \geq 823$ であるときパターン c はパターン c_1, c_2 に**分割可能**であると定義する.

ここで**分割可能**なパターンの 1 例を表.4.2 に示す. 表中のパターン c_1 はパターン c_2 から c_5 のどれかに**分割可能**である. よってこのパターン c_1 は分枝限定法における節点として考える必要がない.

表 4.2: 分割可能なパターンの例

パターン	ビン	余り	ピース 1	ピース 2	ピース 3	ピース 4	ピース 5	ピース 6	ピース 7	ピース 8	ピース 9
c_1	5486mm	286mm	2000mm	400mm	400mm	400mm	400mm	400mm	400mm	400mm	400mm
c_2	4267mm	267mm	2000mm	400mm	400mm	400mm	400mm	400mm			
c_3	3657mm	57mm	2000mm	400mm	400mm	400mm	400mm				
c_4	3048mm	248mm	2000mm	400mm	400mm						
c_5	2438mm	38mm	2000mm	400mm							

4.4.2 分割可能なパターンの削除

分割可能なパターンは分枝限定法の探索木における節点として考え無くても良いため, 入力から生成されるパターンの集合から**分割可能**なパターンを取り除く.

探索木の節点としては考えないが, 分割可能なパターンを後から復元するために分割したパターンを保存しておく. あるパターン c が c_1 と c_2 に分割可能であるとき, 探索木からパターン c を取り除くとともにパターン c_2 を集めておく. これらの c_2 からなる集合を R とする.

$$R = \{c_2, c_2^2 \mid \exists c, c \text{ は } c_1 \text{と } c_2 \text{に分割可能 } \}$$

4.4.3 分割可能なパターンの判定アルゴリズム

パターンを特殊なビットベクトルとして表現し**分割可能**であるかの判定を高速化した. 入力のビンとピースから生成されるパターンの集合が C とする. パターン C の要素であるパターンの $|c|$ が最大であるものを $max(C)$ とする. $max(C) * |P|$ の長さの整数型変数を用意する.

1 種類のピースのビットベクトルを求める関数 $recnum$ を次のように定義する.

$$recnum(num) = \begin{cases} 0 & (num = 0) \\ \Sigma_{i=1}^{num} 1 << (i-1) & (1 \leq num) \end{cases}$$

パターン c に対するビットベクトルを求める関数 $bitvec$ を次のように定義する.

$$bitvec(c) = \Sigma_{i=1}^{|P|}\ recnum(count(c,i)) << (i * max(C))$$

全てのパターンについてビットベクトルをあらかじめ計算しておく. あるパターン c がパターン c_1 c_2 に**分割可能**であるとき, パターン c_1 , c_2 は C の中に必ず存在する. あるパターン c が c_1 c_2 **分割可能**であるかは C の中に次の条件を満たす c_1 が存在するかを判定すれば良い. $bitvec(c)\ And\ bitvec(c_1) = bitvec(c_1)$ かつ,$bitvec(c1) > bitvec(c2)$ かつ $rest(c) > rest(c1)$ であればパターン c はパターン $c_1,\ c_2$ に**分割可能**である. ここで, パターン c_2 はパターン c からパターン c_1 に含まれるピースを取り除き, 新たなビンを割り当てた物とする.

現実的な入力では整数型の最大ビット長である 64bit を越えることがある. このときは 64bit 型整数の配列として実装する. このとき, 1 種類のピースのビットベクトルの幅を配列の要素間で分断すると処理が面倒となるため, $64\ mod\ (max(C) + o_f) = 0$ となる o_f を見付て $max(C)$ 代わりに用いてビット幅を調整すると良い.

表.4.2 のパターンをビットベクトルで表現したものを表.4.3 に示す. ここでは簡単のため $max(C) = 8$ であり, ピースの集合は $len(p_1) = 2000mm$ と $len(p_2) = 400mm$ である $P = [p_1,\ p_2]$ であると仮定する. 論理積の演算により高速に**分割可能**なパターンを見つけられることが分かる.

表 4.3: ビットベクトル表現の例

パターン	ビン	余り	$recnum(count(c,1))$	$recnum(count(c,2))$	$bitvec(c_i)$
c_1	5486mm	286mm	00000001	**11111111**	**11111111**00000001
c_2	4267mm	267mm	00000001	**00011111**	**00011111**00000001
c_3	3657mm	57mm	00000001	**00001111**	**00001111**00000001
c_4	3048mm	248mm	00000001	**00000011**	**00000011**00000001
c_5	2438mm	38mm	00000001	**00000001**	**00000001**00000001

4.4.4 分割可能の定義を用いた探索

入力のビンとピースからパターンの集合を生成する．生成したパターンの集合から分割可能なパターンを取り除き, これを元に探索木を構成する．探索中に**計算状態**で使用されていないピースの長さの総和が 3048mm 以下となるまで分枝限定法で探索を行う．

使用されていないピースから構成可能なパターンが R の要素であるとき計算状態に存在する分割後のパターンに結合可能なパターンの集合 S ($\subseteq R$) を選び分割されたパターンのにピースを追加し新たなビンを割り当て分割前のパターンに復元する．使用されていないピースから構成可能なパターンが R の要素でないときは通常どおりに子節点を探索する．

定義 4.6 (パターンの統合) *2*つパターン c からパターンの統合しパターン列 c' を作成する処理を次のように定義する.

1. $bin(c) \leq 3048$ であるとき, パターン c に含まれるピースからを新たなビンに詰めて, 新しいパターン c' とする.

2. $bin(c) > 3048$ のとき 2 つのパターのピースの長さの総和以下のビンは無いので,c' を単にパターン列 cc とする.

証明 4.2 (分割可能を用いても最適解が得られ, ピースの結合は高々２回で良い) ある解 $\alpha^* = c_1^{k_1} c_2^{k_2} ... c_l^{k_l}$ が最適解であるとする. c^k はパターン c が k 個連続した列であることを示す. パターン c_i $(1 \leq i \leq l)$ が**分割可能**であるとき, 分割したパターンを $c_{i\,1}$, $c_{i\,2}$ とする. パターン c_i が分割できないときも, 同様に $c_{i\,1} = c_i$, $c_{i\,2} = []$ とする.

α^* は定義を元に**分割可能**なパターンを分割し, パターンの統合を繰り返すとパターン列 EF が得られる. このときパターン列 E, F は次に示すとおりである.

$$E = c'^{\lfloor k_1/2 \rfloor}_{1\,1} ... c'^{\lfloor k_l/2 \rfloor}_{l\,1} c^{\lceil k_1/2 \rceil - \lfloor k_1/2 \rfloor}_{1\,1} ... c^{\lceil k_l/2 \rceil - \lfloor k_l/2 \rfloor}_{l\,1}$$

$$F = c^{k_1}_{1\,2} ... c^{k_l}_{l\,2}$$

また, **分割可能**の定義より c_i が $c_{i\,1}, c_{i\,2}$ に**分割可能**であれば $c_{i\,2}$ は $len(b_0) = bin(c_i) - bin(c_{i\,2}) = \Delta * v$ となる特殊なビン b_0 を持つものとする. ここで,$rest(c_i) \geq rest(c_{i\,1})$ であり,$\Sigma_{r \in piece(c_{i\,2})}\, len(r) \geq 823$ であることより v は $(2 \leq v \leq 6)$ をとる.

パターン列 F' に含まれるパターンを長さが *6096mm* の新たなビンを用意して F に含まれるピースを詰められるだけ全て詰め, このパターン列を F' とする. これらのパターンのピースを Δ の自然数倍の長さのビンに詰める限りは最適性は保存される.

このとき,*6096mm* のビンに詰められなかった余りのピースの長さの総和 f とする.

1. $2438 - \Delta < f \leq 5486$
   f以上かつ最小のビンに詰めて新たなパターン c_r を作成し EF の末尾に追加し

   $$\beta^* \;=\; EFc_r$$

   最適性を保存して変形されたので β^* もまた最適解である.

2. $f \leq 2438 - \Delta$ のとき
   分割可能の定義より残されたパターンは高々*2*個である. この場合最適性を保存しない可能性がありこれらのパターンに含まれるピースを新たなビンにいれることはできないため, 分割前の元のパターンに結合する.

   $$\beta^* \;=\; E'F$$

   最適性を保存して変形されたので β^* もまた最適解である.

このようにして得られた解 β^* もまた最適解であり, **分割可能**な定義に基づいてパターンを削減しても最適解が得られる. また、分割したパターンを元のパターンに結合する作業は高々*2*回で良いことが言える.

4.5 動的ハッシュ法

分枝限定法で探索する過程で得られた部分最適解をハッシュ表へ登録し，再利用することで探索節点を削減するという動的ハッシュ法を用いる．図.4.2 において, ある節点 c の子孫を探索して探索木の葉まで探索して得た S に対する部分最適解がある場合はピースの集合 S からハッシュ値を得てそれキーとしてハッシュ表に部分最適解が得られたと言う情報とともに部分最適解を登録する．ここで S は節点 c を探索した時点で使われていないピースの集合である．もし, 探索を進めて節点 c'のように S に対する部分最適解が必要な場合があればこの節点の探索を省略し, ハッシュ表から得た最適解を**計算状態**に加えバックトラックする．

また, 全ての節点を探索する場な単純な枝刈りのみを行う探索においては, 部分木を探索して得られた解がその部分木の部分最適解となる．しかし, パターンクラスや分割可能の定義などの枝刈りの使用によって探索が省略され実際には葉まで探索するケースが少ない. このため, 部分木を探索しても部分最適解が得られないことがある．そこで, ある部分木と**計算状態**の組み合わせで部分最適解が得られないことが分かったときは**計算状態**のコストと部分最適解を得られなかったと言う情報を枝刈りのための情報としてハッシュ表に登録する．同様に S をキーとしてハッシュ表を引いたとき格納されている情報がコストであるときは, 現在の**計算状態**のコストが大きい場合は前回より小さなコストでも部分最適解が得られなかったことを根拠に探索を省略する．

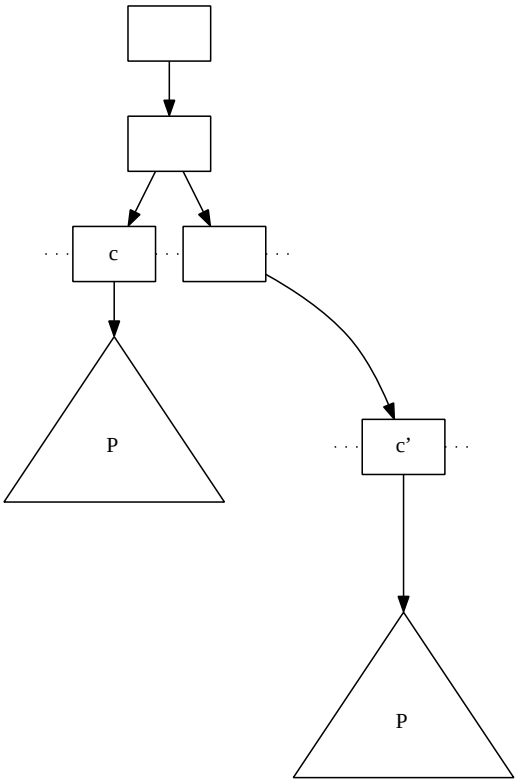

図 4.2: 動的ハッシュ法

ハッシュ表へ**部分解**を登録するために**部分解**に含まれるピースの集合を 2 進数列で表現し, 部分解のハッシュ値とする．

定義 4.7 (ピースの集合の符号) 整数を 2 進数で表現したときのビット幅を返す関数を bit, 各 j $(1 \leq j \leq |P|)$ に対して $shift(j) = \Sigma_{i=j}^{|P|}\ bit(num(p_i))$ とする．このとき，ハッシュ値を生成する関数を $key(\alpha) = \Sigma_{i=1}^{|P|}\ num(\alpha,\ i) * 2^{shift(i)}$ とする．

4.5.1 動的ハッシュ法のアルゴリズム

ある節点の**計算状況**を γ とする．γ 中に含まれるピースの集合を S_1 とし，入力のピースの集合から S_1 を除いた集合を S_2 とする．ハッシュ表 $Hash$ には（S_2 に対する最適解の有無，パターンの列）という形の値を格納する．ここでパターンの列は部分最適解が存在するときには S_2 の部分最適解であり，最適解が得られた値を登録する時に着目している節点の**計算状態** γ である．ハッシュ表に値が既に登録されているときは次の手順でハッシュ表を利用または更新する．

- $Hash[key(S_1)]$ が部分最適解を持つならば，その値を利用する
- $Hash[key(S_1)]$ が部分最適解を持たないならば
  - $cost(Hash[key(S_1)][1]) \leq cost(\gamma)$ ならば子節点の探索を省略する
  - $cost(Hash[key(S_1)][1]) > cost(\gamma)$ ならハッシュ表に次の値を登録する

着目した節点からの探索において

- 葉まで調べるパスが存在しないとき $Hash[key(S_1)] =$ (最適解無し, γ)
- 葉まで調べるパスが存在する $Hash[key(S_1)] =$ (最適解有り, S_2に対する最適解)

表 4.4: 分割可能な定義によるパターンの削減

入力例	$\|P\|$	sum(P)	分割処理前	分割処理後
1	11	83	83	63
3	16	197	108	78
4	6	161	13	9
5	5	73	23	12
6	7	57	59	33
7	12	91	204	95
8	5	72	6	6
9	10	91	129	95
10	9	76	42	28
11	7	65	22	11
12	7	78	35	19
13	8	125	64	38
14	7	79	59	26
15	6	91	20	20
16	5	86	11	8
17	9	64	46	43
18	8	63	26	24
19	8	66	53	28
20	8	68	27	21
21	5	56	8	8
22	3	58	7	6
23	8	66	24	13
24	5	72	11	8
25	16	118	-	-
26	19	65	604	253
27	53	228	-	-
28	20	72	-	-
29	90	330	-	-
30	67	434	-	-
31	51	606	-	-
32	38	259	-	-
33	16	36	415	268
34	14	408	-	-
35	37	650	-	-
36	50	458	-	-
37	24	116	-	-
38	24	177	-	-
39	82	332	-	-
40	36	107	-	-
41	25	314	-	-
42	43	403	-	-
43	25	95	-	-
44	14	114	-	-
45	49	461	-	-
46	21	226	-	-
47	29	220	-	-
48	8	142	-	-
49	13	152	-	-
50	91	263	-	-

第5章　実装と実験結果

本研究において提案した新しいアルゴリズムを C#言語を用いて実装しその計算時間を計測した．Intel Core2Quad プロセッサ Q9450(2.66GHz) を搭載したコンピュータを用いて野呂のアルゴリズムと同様の実装と計算時間を比較した．ここでは株式会社鈴工より提供された木造住宅を建築する際に切り出すときに実際に必要とされるビン（材木）とピース（柱）のデータを使用した．提供されたデータ中のピースの種類と総数の平均値はそれぞれ 23 種類と 175 個であった．

ビンは 7 種類あり長さはそれぞれ 6096mm，5486mm，4876mm，4267mm，3657mm，3048mm，2438mm のものである．

従来手法との比較を表.5.1 に示す. また, 提案した動的ハッシュ法と分割可能の定義の効果の違いについての実験結果を表.5.2 に示す.

5.1　従来手法との比較

木造住宅の実際の建築に使われる入力例 49 例の中で 25 例について最適解を求めることが出来た．また，最適解を求めることができた 25 例中，12 例については計算時間が改善し特に 3 例については目覚ましい改善結果となり，本研究における提案手法の有効性を実証した．

現実的な時間で計算可能な入力に対しては計算時間を大幅に改善した．ただ, 現実的な時間で計算できない入力を計算できるほどには改善できなかった．現実的な時間で計算できなかった残りの 24 例については近似アルゴリズムを用いて計算する必要がある．

5.2　動的ハッシュ法と分割可能の比較

大幅な改善が見られた入力 7,9,26 である. このうち入力 7 と 26 については動的ハッシュ法により大幅に改善している. 入力 9 は分割可能な定義によるパターンの削減により大幅な改善が見られる.

表 5.1: 従来手法との比較

			提案手法		従来手法	
入力例	$\|P\|$	sum(P)	探索節点数	探索時間	探索節点数	探索節点数
1	11	83	24769	0.5	91807	0.5
3	16	197	9004	1.5	354480	1.9
4	6	161	167	0.5	577	0.5
5	5	73	94	0.2	23762	0.2
6	7	57	408	0.2	1434	0.2
7	12	91	343628	4.9	394719936	675.0
8	5	72	75	0.2	142	0.2
9	10	91	491704	7.3	62516144	105.9
10	9	76	2156	0.3	72225	0.4
11	7	65	71	0.2	27297	0.3
12	7	78	346	0.2	3305770	4.4
13	8	125	31988	0.8	7619029	9.2
14	7	79	226	0.3	6775	0.3
15	6	91	684	0.2	1302	0.2
16	5	86	97	0.2	8258	0.2
17	9	64	3548	0.2	90060	0.4
18	8	63	143	0.3	477	0.3
19	8	66	876	0.3	97488	0.4
20	8	68	1302	0.2	45409	0.3
21	5	56	50	0.1	99	0.1
22	3	58	379	0.05	296	0.05
23	8	66	143	0.3	1680	0.3
24	5	72	78	0.2	2082	0.2
25	16	118	-	-	-	-
26	19	65	35600	1.0	127946915	218.4
27	53	228	-	-	-	-
28	20	72	-	-	-	-
29	90	330	-	-	-	-
30	67	434	-	-	-	-
31	51	606	-	-	-	-
32	38	259	-	-	-	-
33	16	36	5376	0.3	173672	0.9
34	14	408	-	-	-	-
35	37	650	-	-	-	-
36	50	458	-	-	-	-
37	24	116	-	-	-	-
38	24	177	-	-	-	-
39	82	332	-	-	-	-
40	36	107	-	-	-	-
41	25	314	-	-	-	-
42	43	403	-	-	-	-
43	25	95	-	-	-	-
44	14	114	-	-	-	-
45	49	461	-	-	-	-
46	21	226	-	-	-	-
47	29	220	-	-	-	-
48	8	142	-	-	-	-
49	13	152	-	-	-	-
50	91	263	-	-	-	-

表 5.2: 動的ハッシュ法及び分割可能に基づく手法の比較

			動的ハッシュ及び分割可能		分割可能のみ		動的ハッシュのみ	
入力例	$\|P\|$	sum(P)	探索節点数	探索時間	探索節点数	探索時間	探索節点数	探索時間
1	11	83	24769	0.5	30269	0.5	28271	0.5
3	16	197	9004	1.5	17419	1.7	79281	2.9
4	6	161	167	0.5	276	0.5	236	0.5
5	5	73	94	0.2	94	0.2	1954	0.2
6	7	57	408	0.2	464	0.2	531	0.2
7	12	91	343628	4.9	682103	7.7	51833664	767.0
8	5	72	75	0.2	84	0.2	75	0.2
9	10	91	491704	7.3	14440239	252.5	591220	9.3
10	9	76	2156	0.3	10508	0.4	22350	0.4
11	7	65	71	0.2	71	0.2	5715	0.3
12	7	78	346	0.2	1240	0.2	136547	0.9
13	8	125	31988	0.8	285210	1.4	35168	0.9
14	7	79	226	0.3	226	0.3	3541	0.3
15	6	91	684	0.2	4790	0.2	684	0.2
16	5	86	97	0.2	97	0.2	921	0.2
17	9	64	3548	0.2	29583	0.3	3549	0.2
18	8	63	143	0.3	191	0.3	144	0.3
19	8	66	876	0.3	951	0.3	8642	0.3
20	8	68	1302	0.2	9229	0.2	12120	0.3
21	5	56	50	0.1	50	0.1	50	0.1
22	3	58	379	0.05	379	0.05	379	0.05
23	8	66	143	0.3	145	0.3	270	0.3
24	5	72	78	0.2	78	0.2	526	0.2
25	16	118	-	-	-	-	-	-
26	19	65	35600	1.0	53910	1.1	8667361	171.6
27	53	228	-	-	-	-	-	-
28	20	72	-	-	-	-	-	-
29	90	330	-	-	-	-	-	-
30	67	434	-	-	-	-	-	-
31	51	606	-	-	-	-	-	-
32	38	259	-	-	-	-	-	-
33	16	36	5376	0.3	9626	0.4	7026	0.4
34	14	408	-	-	-	-	-	-
35	37	650	-	-	-	-	-	-
36	50	458	-	-	-	-	-	-
37	24	116	-	-	-	-	-	-
38	24	177	-	-	-	-	-	-
39	82	332	-	-	-	-	-	-
40	36	107	-	-	-	-	-	-
41	25	314	-	-	-	-	-	-
42	43	403	-	-	-	-	-	-
43	25	95	-	-	-	-	-	-
44	14	114	-	-	-	-	-	-
45	49	461	-	-	-	-	-	-
46	21	226	-	-	-	-	-	-
47	29	220	-	-	-	-	-	-
48	8	142	-	-	-	-	-	-
49	13	152	-	-	-	-	-	-
50	91	263	-	-	-	-	-	-

第6章　まとめ

本研究ではまず野呂の高速最適化アルゴリズムにおいて提案されたパターンクラスの概念をよりすることによりパターンクラス数を減らすことに成功し，分枝限定法における探索節点数が削減できることを実証した．次に，分枝限定法で求めた部分最適解をハッシュ表に保存して再利用する動的ハッシュ法を適用した．また，部分最適解が得られなかった場合にはハッシュ表に枝刈り可能な情報を格納し，以後，その値を利用して枝刈りを行うようにアルゴリズムを改善した．さらに，入力から生成されるパターンに対して分割可能なパターンという概念を導入して，パターンの総数及びその探索範囲を大幅に削減した．これらの新しい手法を導入することにより野呂らの高速最適化アルゴリズムを高速化すると共に，現実的な時間で最適解を計算可能な入力のサイズを広げた新しいアルゴリズムを提案した．

本研究では提案した手法の有効性を確認するために，本アルゴリズムを実装して実際の現場で使われているデータ（49 例）を用いて先行研究のアルゴリズムと計算時間と探索接点数を比較する実験を行った．49 例中 13 例に改善が見られた．これらの実験結果より大幅な改善を確認することができ，本手法の有効性を実証した．

残された課題としては一般化したパターンクラスにおける包含関係の計算を近似アルゴリズムではなく高速に計算可能な最適化アルゴリズムに置き換えることや, 分割可能な定義をより一般化すること, 2 段階に分割可能な場合など分割可能な定義における証明が不十分であるのでより詳細に証明すること, また, 本研究の提案手法で高速化した最適化アルゴリズムを野呂の高精度近似アルゴリズムに組み込み近似度が改善できるか実験により確かめることや, 高速化した利点をいかして更なる改善を行うことが考えられる．

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通夫教授，三橋一郎助教，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落合美子事務員、所属研究室の皆様に深く感謝の意を表します.

参考文献

[FL86] D. K. Friesen and M. A. Langston. Variable sized bin packing. In *SIAM J. COMPUT Vol.15*, pp. 222–229, 1986.

[野呂 10] 野呂耕三. 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. 修士論文, 三重大学大学院, 2010.