工業的制約付き二次元ビンパッキング問題に対する最適化手法の提案

Local Search Algorithms for Two-Dimensional Bin Packing Problem with Industrial Constraints

情報工学専攻　久武　優一
Information and System Engineering HISATAKE Yuichi

1. はじめに

二次元ビンパッキング問題とは，与えられた二次元領域に対して重ならないように複数の長方形形状の製品を詰め込む組合せ最適化問題である．材料の削減，輸送コストの削減，製品の小型化など様々な分野での応用が考えられるが，一般的に NP 困難であるためアルゴリズムの設計は極めて重要である．本研究ではガラスや鋼材などの切り出し問題において現れる様々な工業的制約が追加された問題を扱う．

2. 問題説明

本研究で扱う問題は，ROADEF が 2018 年のコンペティションで出題した順序制約付きの二次元ビンパッキング問題である[1].

2.1. 用語説明

製品の切り出し元となる定型の長方形領域を母材と呼ぶ．まだ切り出しが終わっていない領域のうち，母材でないものを中間材と呼ぶ．母材または中間材を辺に対して平行かつ端から端まで切ることをギロチンカットと呼ぶ．また，母材を鉛直方向にいくつかに切ることを 1-cut とし，切りだしたそれぞれの中間材をさらに水平方向にいくつかに切ることを 2-cut とする．以下同様に鉛直方向と水平方向に交互に切ることをα-cut（α=3,4,...）とする（図 1 参照）．また，このαをカットのステージと呼ぶ．各母材には，製品に含まれてはならない領域があり，これを不良領域と呼ぶ．使用した母材のなかで，切り出しが終わったときに，どの製品にも使われていない領域を破棄領域と呼ぶ．ただし，最後に使用した母材で，最後の 1-cut で切り出される破棄領域を特別に再利用可能領域と呼ぶ．製品はいくつかの独立な順序つき集合に分けられており，この集合をスタックと呼ぶ.

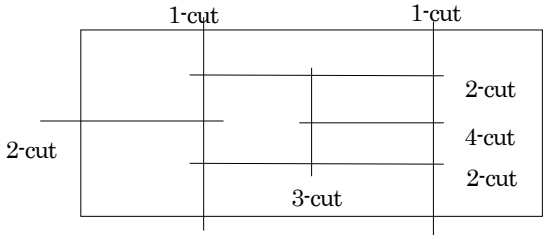

図 1 α-cut の例

2.2. 目的関数および制約条件

再利用可能領域を除いた破棄領域の最小化を行う．与えられた母材から全ての製品を切り出す時に，製品を互いに重ならないように配置する(本研究では配置制約と呼ぶ)などの二次元ビンパッキング問題の標準的な制約の他に，本研究では以下に述べる制約も考慮する.

各スタック内の製品は順番に切り出されなければならない，これを順序制約と呼ぶ．なお切り出される順番は同じ中間材の中で奇数ステージでは左から，偶数ステージでは下からの順番になる． 図 2 の配置では左は順序制約を満たすが，右の配置は赤字で示した製品が順序制約を満たしていない.

3-cut 以内の切り出し，もしくは 3-cut のときに現れる中間材をただ 2 つに分ける特別な 4-cut のみを許容する，これを 3 ステージカット制約と呼ぶ.

製品を切り出す際は不良領域を含んではならず，ギロチンカットによって不良領域を切断してはいけない．これらをまとめて不良領域制約と呼ぶ．不良領域制約の例を図 3 に示す.

それぞれのカットによって生じる中間材及び製品の幅はカットのステージごとに上下限の制約がある，これをカット幅制約と呼ぶ.

※a_iはスタックaのi番目の製品を表す

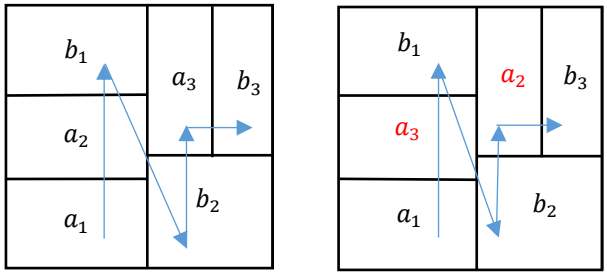

図 2 順序制約を満たす配置(左)と満たさない配置(右)

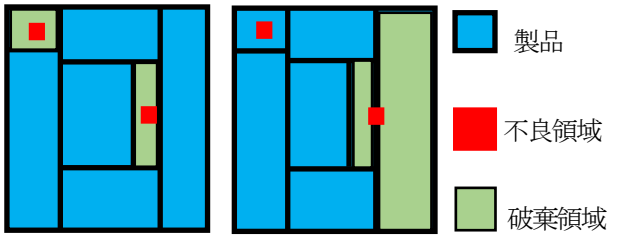

図 3 不良領域制約を満たす配置（左）と
満たさない配置（右）

3. 木構造による解表現

ギロチンカットによって得られる長方形配置は，図 4 のようにノードや親子関係を構成することによって，多分木で表現できる[2]. 具体的には以下のように構成する. 母材の全体集合を表すノードを木の根とし，これを全体ノードと呼ぶ. 全体ノードは母材を表すノードを使用した順番に左から子のノードとして持ち，これらのノードを母材ノードと呼ぶ. それぞれの母材ノードは 1-cut によってできる中間材を子に持ち，切り出される順番に左から並ぶ. 以下同様に，次のステージの中間材または製品を表すノードを子に持ち，製品を表すノードは全て葉となる. 全体ノードを除く全てのノードは，実際に配置する左下の座標とそのノードが表す領域の大きさを持つ. 各制約の確認および目的関数の計算は，製品数をnとすると，$O(n)$ 時間でできる. この木は以下の特徴を持つ.

- 深さ優先探索のラベリングは, 製品が切り出される順番に対応する.
- 各母材ノードを根とする部分木の各ノードの深さは，各ノードに対応する中間材及び製品が切り出された時のカットのステージ数に対応する.

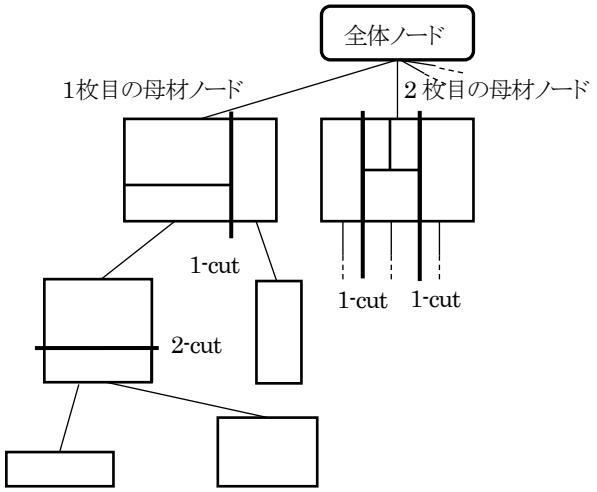

図 4 木構造による解表現

4. 提案アルゴリズム

貪欲法で初期解構築を行ったのち，構築した解を初期解として局所探索法を用いて改善する.

4.1. 貪欲法を用いた解構築

本問題は順序制約が非常に厳しく，実行可能解が標準的な二次元ビンパッキング問題と比べて少ない. このため初期解構築の段階でこの制約を強く意識する必要がある. また，3 ステージカット制約により，製品二つに別れる様な特殊な 4-cut がなければ，2-cut でできる中間材の中には図 5 のように一次元に製品が並んでおり，一次元のビンパッキング問題を再帰的に解くような問題と解釈することもできる. そこで,各スタックの先頭の製品のうち，現時点での充填率がなるべく高くなるように製品の部分集合を選び，その部分集合の集合，そのまた集合を充填率が良くなる様にパッキングしていく. このようにすることで，全ての制約を満たした解を効率よく見つけることができる.

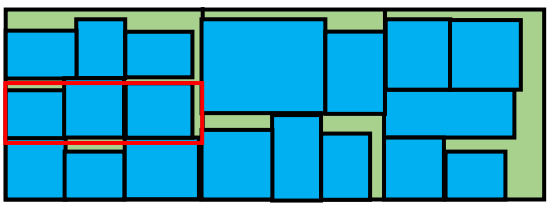

図 5 全体の配置と一次元パッキング領域(赤枠)

まず，母材を切る幅を固定して，その幅の中間材に収まるように各スタックの先頭から製品を取り出し，左から並べていくことを考える. このように製品を詰め込むと,配置制約と順序制約を満たせるだけでなく，

ちょうど 3-cut もしくは制約遵守の 4-cut の配置になり 3 ステージカット制約も満たせる．2-cut の幅は，カット幅制約により一番高い製品と二番目に高い製品によって決まるので，並べた製品の高さがその 2-cut の幅に近ければ充填率はよくなる．図 6はある幅と高さの領域に対して，各スタックの先頭から製品を取り出し並べるイメージである．取り出す製品の高さの範囲を固定し，その高さに近い製品を詰め込むことを，その都度配置をリセットして様々な高さで繰り返す．このうち，1 番充填率の良かった高さを採用し，2-cut の位置を仮決めして，残りの領域でこれを繰り返す．残りの領域に製品を置けなくなったら一度リセットし，今度は別の 1-cut の幅で母材を切り，これらを繰り返す．2-cut の時と同じように，1 番充填率の良かった幅で 1-cut の幅を決定し，これまでの配置を確定させる．これを全ての製品が詰め込まれるまで繰り返す．ただし，母材の残りの幅および残りの高さが少なくなったときに同じ方法で詰めていくと半端な領域が余ってしまい無駄になってしまう可能性がある．そのため，残りの幅および残りの高さが一定以下のときは，その残りを 2 分割してそれぞれに詰め込みを行い，最も良い分割を求める．

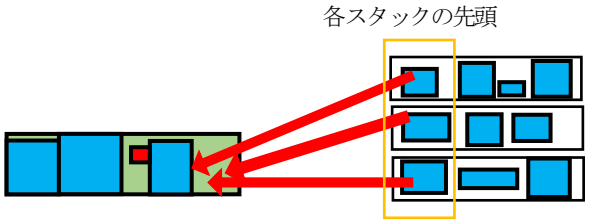

図 6 詰め込みのイメージ

4.2. 近傍設計

製品や中間材の挿入，交換は対応するノードを根とする部分木の挿入，交換に対応する．従って，子孫関係のないノードまたは挿入部分の組を作り挿入および交換を行うことを近傍操作とする．ただし近傍操作は図 7 のような単純に部分木の挿入をするだけの近傍と，図 8 および図 9 のように，リーフとして存在するノードと挿入ノードの二つを子ノードとしてもつ中間材ノードを新たに作り再構成する近傍の 2 種がある．交換近傍の例は図 10 に示す．

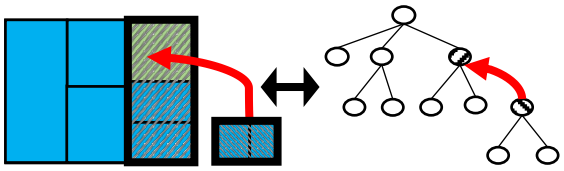

図 7 着目ノードに子ノードとして挿入する近傍の例

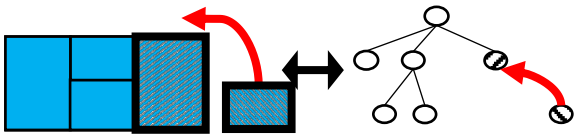

図 8 リーフを兄弟ノードとして再構成する近傍の例

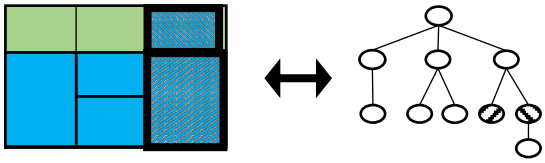

図 9 図 8 の再構成後

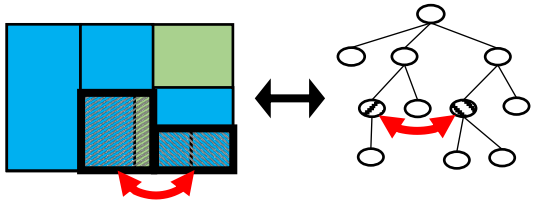

図 10 交換近傍の例

4.3. 近傍解に対する制約確認

本研究では，現在の解と近傍解の類似性を活用し，各ノードに対して以下の情報を前処理や探索中に付与することで，近傍解における各制約の確認を高速に行えるようにする．

配置制約および不良領域制約の確認

まず，各ノードが不良領域によって動きがどれだけ制限されるかを表す可動範囲をボトムアップで計算する．次にこれを用いて，配置制約および不良領域制約を満たしながら，どこまで自身の領域を大きくすることができるか表す最大伸長をトップダウンで計算する．これを用いることで，ノードの挿入や異なる母材間のノードの交換における配置制約および不良領域制約が近傍解一つあたり $O(1)$ で判定できる．可動範囲および最大伸長の前計算は全体で $O(n)$ にて計算できる．

順序制約の確認

各ノードに，製品を表すノードなら対応する製品の属するスタックと番号，母材もしくは中間材を表すならどのスタックの製品がスタック内の順序で何番目から何番目まで配置されているかを覚えておく．これをスタック情報と呼ぶ．なお，この情報は深さ優先後行順に決定できるので，移動先と移動元の組を作るときに自然に計算できる．この情報を用いると，順序制約を $O(1)$ で判定できる．また，一度制約違反を見つければ，深さ優先順でそれより先の移動先との組は全て制約違反となるため，それ以降の移動先は調べなくても良くなり，調べる候補を削減できる．例えば，図 11 のように，各ノードにスタック情報を持たせており，赤矢印がさすノードと青矢印がさすノードを入れ替えることを考える．図のようなノードの入れ替えを行うと，オレンジ色で塗り潰したノードを赤矢印がさすノードが追い越し，スタック a で順序制約を違反する．従って，交換操作の一方（赤矢印のノード）を固定して，もう一方（青矢印のノード）を深さ優先後行順で動かしている場合，オレンジ色のノードで順序制約違反を見つけた時に，それより先を調べる手間が省ける．

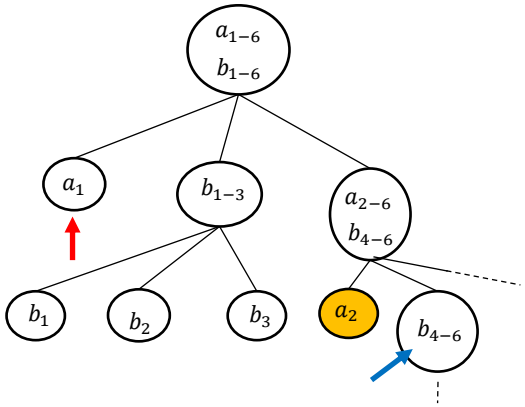

図 11 順序制約の確認を省略できる例

3 ステージカット制約の確認

各ノードに部分木の高さを覚えておけば 3 ステージカット制約は $O(1)$ で判定できる．前計算は全体で $O(n)$ にて計算できる．

4.4. 局所探索のための評価関数

本問題の目的関数は最後に切り出される製品の位置が改善されないと値が変わらない．従って，目的関数値は同じでも，改善に向かっていることを評価できる関数を設計しなければならない．本研究では近傍設計で着目している中間材の充填率を評価関数とし，各近傍操作の対応するノードの親ノードが表す中間材の充填率が改善された時に移動する．

4.5. 近傍操作後の再計算

近傍解へ移動した後の座標，大きさ，可動範囲，最大伸長，部分木の高さの再計算はいずれも最悪の場合 $O(n)$ 時間かかるが，変更の可能性のあるノードだけ計算することで，現実的には効率化できる．

4.6. 実験結果

以下の実験環境で，出題元の[1]にあるデータセットに対して数値実験を行なった．なお，局所探索によって改善した解がなかったため，これらの結果は全て初期解の時点での値である．なお評価値は比較しやすくするため，充填率を掲載する．充填率を最小化することは元の目的関数を最小化することと同じである．

表 4.1 実験結果の一部

問題	製品数	スタック数	充填率	時間[秒]
A1	5	1	2569766	63.7%
A2	72	72	29678309	72.2%
A5	97	12	13446513	80.8%
A15	392	14	26451971	90.0%
B7	241	241	36854199	83.6%

得られた考察として以下があげられる

- 貪欲法では，製品の大きさを考慮していないため扱いにくい大きい製品が後ろの方に配置されてしまい，後ろの母材の充填率が悪い．
- 順序制約や配置制約の厳しさから現在の近傍設計では，ある程度良い解から動くことが難しい．

参考文献

[1] [online] http://www.roadef.org/challenge/2018/en/index.php (2019/12/26 アクセス)

[2] R. Otten, Automatic Floorplan Design, Proceedings of the 19th Design Automation Conference (1982), p.261–267.