数理解析研究所講究録
第1981巻 2016年 5-26

ラスタ表現された図形の詰込み問題に対する局所探索法

大阪大学・大学院情報科学研究科・情報数理学専攻 村上 祥平
Department of Information and Physical Sciences,
Graduate School of Information Science and Technology, Osaka University
大阪大学・大学院情報科学研究科・情報数理学専攻 梅谷 俊治
Department of Information and Physical Sciences,
Graduate School of Information Science and Technology, Osaka University
NTTデータ数理システム 中野 雄介
NTT DATA Mathematical Systems Inc.
大阪大学・大学院情報科学研究科・情報数理学専攻 森田 浩
Department of Information and Physical Sciences,
Graduate School of Information Science and Technology, Osaka University

1 はじめに

詰込み問題とは, いくつかの図形を互いに重ならないように与えられた領域内に配置する問題である. この問題は, 図形の次元や形状により様々なバリエーションを持ち, 製造業や物流業などの多くの分野に応用を持つ問題である.

近年, 工業上の応用事例において, 様々な曲線で描かれた図形を詰込み問題で取扱う必要が生じている. 図形の表現方法は, 大きく分けてベクタ形式とラスタ形式の2種類が知られている. ベクタ形式とは点と線分で図形を表現する方法である. この形式では図形対の重なり判定を各線分が交点を持つか否かで判定するが, 曲線など複雑な形状の図形の重なり判定では, 直線と曲線や曲線同士の交点の有無を判定するため, 重なりを高速に判定することは容易ではない. 一方で, ラスタ形式とは画像を色のついたドットと呼ばれる点集合で図形を表現する方法である. 重なり判定はドット同士の重なりの有無で判定するため, 曲線など複雑な形状の図形の重なり判定においても, ドット同士の重なりの有無で判定できる.

ラスタ表現された図形の詰込み問題を解くデータ構造とアルゴリズムは提案されているが, 実用的な規模の例題に対して現実的な計算時間で十分な精度の解を得られていない [3, 14, 15]. 本研究では, ラスタ表現された図形の詰込み問題に対して効率的な局所探索法を提案する.

2節では, 先行研究の手法について述べ, 先行研究の課題と本研究の目的を示す. 3節では, 図形の表現方法について述べる. 4節では, 本研究で扱うストリップパッキング問題と部分問題の重なり度最小化問題の定式化を行う. 5節では, 図形対の重なり判定に用いるミンコフスキー差について述べる. 6節では, アルゴリズムの概要について述べ, 7節では, 初期解の構築方法について述べる. 8節では, 重なり度最小化問題を解くために用いる局所探索法について述べ, 9節では, 数値実験の結果と考察について述べる. 最後に, 10節では, 結論について述べる.

2 研究背景と目的

ストリップパッキング問題とは, 布地や金属版など母材が十分な幅を持ち, 必要となる母材の幅を最小化する, 詰込み問題の一種である. この問題は, 図形の集合と幅が固定され, 長さが可変である長方形領域が与えられる. 全ての図形が重ならないように長方形領域内に配置するという制約の下で, 長方形領域の長さを最小化することが目的である. この問題は図形の回転に依存して 3 つのバリエーションを持つ. (1) 任意の回転角を許す, (2) 有限個の回転角を許す, (3) 回転を許さない場合が存在し, 本研究では (2) の場合を扱う. この問題は繊維産業において布の使用量を最小化する事例などに応用されており, コンピュータの性能向上とアルゴリズムの高速化により, 実用的な規模の例題に対して良い解が得られるようになっている.

図形の表現方法は, 大きく分けてベクタ形式とラスタ形式の 2 種類が知られている. ベクタ形式とは点と線分で図形を表現する方法である. 従来の研究では, 図形の表現方法としてベクタ形式が用いられる場合が多く, ベンチマーク問題例に対する数値実験結果では No-Fit Polygon と呼ばれるデータ構造を使用した方法が良い結果を出している. 一方で, No-Fit Polygon にはいくつかの問題点が存在する. No-Fit Polygon は穴, 凹面を含む図形, ジグソー型の図形を扱う場合, 縮退を引き起こすため実装の際には多くの例外処理が必要となる [5]. また, No-Fit Polygon は直線で表現された多角形を扱う場合には効率良く計算できるが, 一般的な曲線で表現された図形の No-Fit Polygon を高速で作成することは難しい. そのため, 曲線で表現された図形を扱う場合には直線に近似して円弧を含む図形を扱う. 曲線を多数の直線で近似すると図形が実際の形状に近づく. 一方で, 近似に用いる直線の数を増やせば計算時間は増加し, 図形を実際の形状に近づけることと計算時間はトレードオフの関係である. そのため, ベクタ形式では曲線など複雑な形状の図形の重なりを高速に判定することは容易ではない.

ベクタ形式を用いた多角形詰込み問題に対するアルゴリズムについて紹介する. ヒューリスティックを使用した手法として, 図形を領域内に 1 つずつ配置する際に, 配置座標の決定を効率化するアプローチが知られている. Albano ら [1] は, すでに配置済みの図形の右端に接する配置の中で, 可能な限り左側に配置する手法を提案している. また, No-Fit Polygon を利用して図形対が接する配置を発見することで, 手続きの効率化を行っている. No-Fit Polygon は 2 つの図形対が重なりなく接するような配置の集合と定義される [2]. Blazewicz ら [6] は Albano ら [1] の手法を 詰込まれた図形により形成された穴に配置できるように拡張した Bottom-Left-Fill Algorithm を提案している. Gomes ら [10] は No-Fit Polygon を用いて領域内の図形が重ならない配置の中で, 可能な限り左側に配置する手法を提案している. また, 局所探索法を適用することで図形の配置順を効率良く探索している. Burke ら [7] は円弧や穴を含む図形を扱えるように拡張した Bottom-Left-Fill Algorithm を用いて, 円弧や穴を含む図形を扱う手法を提案している.

他のヒューリスティックを使用した手法として, 部分問題を解く手法が知られている. 部分問題として (1) 重なり度最小化問題, (2) コンパクション問題, (3) セパーション問題が知られている. 重なり度最小化問題とは, 与えられた幅, 長さの長方形領域の中に配置するという制約の下で, 図形対の重なりを最小化する問題である. コンパクション問題は与えられた実行可能解において, 長方形領域の長さが最小になるように図形を再配置する問題である. セパレーション問題は与えられた実行不可能解において, 重なりを解消するのに必要な最小移動距離だけ移動させることで実行可能解を得る問題である. Bennell ら [4] はコンパクションアルゴリズムとセパレーションアルゴリズムを組み込んだタブー探索を提案している. Gomes ら [11] はコンパクションアルゴリズムとセパレーションアルゴリズムを組み込んだ焼きなまし法を提案している. Egebald ら [9] は重なり度最小化問題を解くことで多角形ストリップパッキング問題を解くヒューリスティックアルゴリズムを提案している. 図形対が重なっている面積を重なり度としており, 局所探索法の近傍は探索を行

う図形の配置に対して水平方向, 垂直方向に移動させて得られる配置としている. Imamichi ら [12] は, 図形対の重なりを解消するのに必要な最小移動距離を重なり度として, 非線形計画法の考えを組み込んだ局所探索法を繰り返すことで, 重なり度最小化問題を解くアルゴリズムを提案している. Umetani ら [16] は Egeblad ら [9], Imamichi ら [12] の手法をベースとして, 図形対の重なりを解消するのに必要な最小移動距離を重なり度としている.

一方, ラスタ形式とは画像を色のついたドットと呼ばれる点集合で図形を表現する方法である. 重なり判定はドット同士の重なりの有無で判定するため, 曲線など複雑な形状の図形の重なり判定においても, ドット同士の重なりの有無で判定できる. これまで, ラスタ表現された図形の詰込み問題を解くデータ構造はいくつか提案されている. Olveira ら [14] は図形が配置されているピクセルを 1, 配置されていないピクセルを 0 と表現する手法を提案している. 行列によりレイアウトを表現し, 行列の要素値が 1 より大きければ図形が重なっていることが分かる. Segenreich ら [15] は, 図形の重なりだけではなく, 図形対が接触している点を発見できる手法を提案している. 図形の境界上を 1, 図形の境界に囲まれた内部を 3, 図形が重なっていることを 4 以上で表現する. Babu ら [3] は, 図形が配置されているピクセルを 1 以上, 配置されていないピクセルを 0 として表現する手法とは異なる手法を提案した. 図形を配置する領域の境界上または境界の外側は 1 以上の値, 境界の内側は 0 と表現しており, 最も右のピクセルを 1 として, 右から左に 1 ピクセル移動するごとに値を 1 加えている. 図形の表現方法も類似している. 図形の境界の外側を 1 以上の値, 境界上または境界の内側を 0 と表現しており, 同様に最も右のピクセルを 1 として, 右から左に 1 ピクセル移動するごとに値を 1 加えている. 図形を配置した際には, 図形の領域と図形のピクセル値が 0 であるときのみ 1 以上の値となり, それ以外の場合は 0 と表現することで, 配置を表現している.

ラスタ形式を用いた詰込み問題に対するアルゴリズムについて紹介する. Olveira ら [14] はラスタ表現された図形に対する焼きなまし法を提案している. Segenreich ら [15] は重なりを持つ図形に対して, 水平方向に各配置の重なりの有無を確認し, 重なりが無い配置を探索する方法を提案している. 水平方向に探索して, 重なりが無い配置が見つからなければ, 現在の図形の配置から決められた長さだけ上に位置する配置に対して, 水平方向に各配置の重なりの有無の確認を繰り返して, 重なりが無い配置を探索する. Babu ら [3] は遺伝的アルゴリズムにより図形の配置順を探索し, Bottom-Left Algorithm により図形の配置座標の決定を効率化した手法を提案している.

このようにラスタ形式を用いた手法は提案されているが, 実用的な規模の例題に対して現実的な計算時間で十分な精度の解を得られていない. 本研究では, ラスタ表現された図形の詰込み問題に対して効率的な局所探索法を提案する. 本研究では, Umetani ら [16] が提案した局所探索法をベースとして, ラスタ表現された図形の詰込み問題に対して局所探索法を提案する.

3 図形の表現方法

コンピュータ上において, 2 次元の図形はベクタ形式もしくはラスタ形式で表現される. ベクタ形式は図形を点と線分で表現しており, ラスタ形式は図形をドットと呼ばれる点集合で表現する. ベクタ形式やラスタ形式で表現された図形の呼び名は様々であるが, 本研究ではベクタ形式で表現された図形をベクタ図形 (図 1), ラスタ形式で表現された図形をラスタ図形 (図 2) と呼ぶ.

従来, 詰込み問題ではベクタ図形が多く用いられてきた. ベクタ図形は点と線分で表現されており, 図形対の重なり判定は基本的に各線分が交点を持つか否かで判定する. そのため, 曲線など複雑な形状の図形の重なり判定は, 直線と曲線や曲線同士の交点の有無を判定することになるため判定が困難である. 直線の重なり判定においても有限精度で線分が記述されているため, 数値誤差により重なり判定を誤る場合がある.

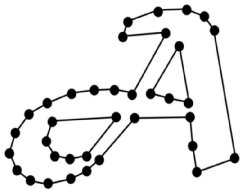

図 1: ベクタ図形

図 2: ラスタ図形

一方, ラスタ図形は点集合で表現されており, 図形対の重なり判定はドット同士の重なりの有無で判定する. ベクタ図形では図形の形状が複雑であるほど重なり判定が困難になるが, ラスタ図形ではどのような図形であってもドット同士の重なりの有無で判定を行うため, 曲線など複雑な形状の図形の重なり判定においても, ドット同士の重なりの有無で判定できる. また, 連続値で表現された線分同士の交点を計算する必要が無く, 数値誤差や縮退を回避するための例外処理を施す必要が無いという利点を持つ. しかしながら, ラスタ図形は点集合で表現されており, 重なり判定にかかる計算量がドットの解像度に依存するという欠点を持つ. また, ドットの数に計算量が依存しているため, 図形対の重なりの有無, 重なりを解消するのに必要な移動距離を容易に得ることはできない. 本研究では 4 節で述べる重なり度の計算において, 図形対の重なりを解消するのに必要な移動距離を重なり度として用いる.

ラスタ図形は各ドットの色データを保持しているが, 本研究では, 図形を走査して有色のドットの区間の始点と終点の座標のみを保持するデータ構造を使用する [13]. 本研究ではこのデータ構造をスキャンライン形式と呼ぶ (図 3). また, 重なりを解消するのに必要な移動距離の計算を効率化するために, 水平方向だけでなく垂直方向にも走査し, 2 つのスキャンライン形式で表現された集合 (以下, スキャンラインの集合と呼ぶ) を作成する.

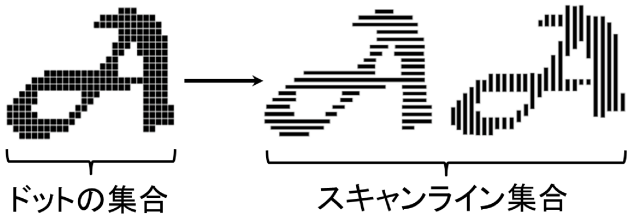

図 3: 水平方向, 垂直方向のスキャンラインの集合

4 定式化

4.1 ストリップパッキング問題

n 個の図形集合 $P = \{P_1, P_2, \ldots, P_n\}$ が与えられるものとする. 図形は許可された回転角の集合 $O = \{O_1, O_2, \ldots, O_n\}$ が与えられる. $O_i (1 \leq i \leq n)$ は図形 P_i が回転可能である角度の集合である. また, 幅 W, 長さ L の長方形領域 $C = C(W, L)$ が与えられる. ここで, $W > 0$ の定数であり, $L > 0$ の変数である. 簡略化のために, $P_i(o)$ は図形 P_i が $o \in O_i$ 度回転していることを表す. 図形 P_i の適当な点を参照点として, 参照点の座標 $\boldsymbol{r}_i = (x_i, y_i)$ により図形 P_i の配置を表現する. 本研究では, 参照点を図形 P_i を囲む Bounding Box の中心とする. 図 4 に参照点の例を示す. ただし, 図 4 の黒い点が参照点である. 本研究ではラスタ図形を扱うため, 各図形 P_i の幅, 長さを

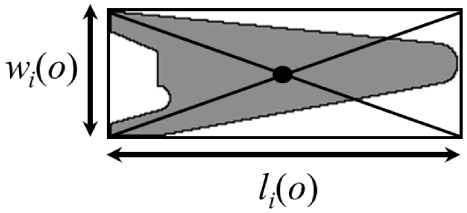

図 4: Bounding Box による参照点の決定

$$w_i(o) = \max\{y \mid (x,y) \in P_i(o)\} - \min\{y \mid (x,y) \in P_i(o)\} \tag{1}$$
$$l_i(o) = \max\{x \mid (x,y) \in P_i(o)\} - \min\{x \mid (x,y) \in P_i(o)\} \tag{2}$$

で表し, 全ての回転角 $o \in O_i$ で $w_i(o) \leq W$, $l_i(o) \leq L$ を満たす. また, $\boldsymbol{r}_i = (x_i, y_i)$ に配置された図形 P_i はミンコフスキー和により表される. ミンコフスキー和を式 (3) に示す.

$$P_i \oplus \boldsymbol{r}_i = \{\boldsymbol{p} + \boldsymbol{r}_i \mid \boldsymbol{p} \in P_i\}. \tag{3}$$

このとき, ストリップパッキング問題は式 (4) のように表される.

$$\begin{aligned}
&\text{minimize} && L \\
&\text{subject to} && (P_i(o_i) \oplus \boldsymbol{r}_i) \cap (P_j(o_j) \oplus \boldsymbol{r}_j) = \emptyset\ ,\ 1 \leq i < j \leq n, \\
& && (P_i(o_i) \oplus \boldsymbol{r}_i) \subseteq C(W,L)\ ,\ 1 \leq i \leq n, && (4) \\
& && o_i \in O_i\ ,\ 1 \leq i \leq n, \\
& && \boldsymbol{r}_i \in \mathbb{R}^2\ ,\ 1 \leq i \leq n, && (5) \\
& && L \geq 0.
\end{aligned}$$

本問題の解は $\boldsymbol{r} = (\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n)$ と $\boldsymbol{o} = (o_1, o_2, \ldots, o_n)$ により表される. 実行可能解 $(\boldsymbol{r}, \boldsymbol{o})$ が与えられた際の長方形領域 C の幅 $L(\boldsymbol{r}, \boldsymbol{o})$ は, 式 (6) のように表される.

$$\begin{aligned}
L(\boldsymbol{r}, \boldsymbol{o}) = \quad & \max\{x \mid (x,y) \in (P_i(o_i) \oplus \boldsymbol{r}_i), 1 \leq i \leq n\} \\
& - \min\{x \mid (x,y) \in (P_i(o_i) \oplus \boldsymbol{r}_i), 1 \leq i \leq n\}.
\end{aligned} \tag{6}$$

図 5 にストリップパッキング問題の実行可能解の例を示す.

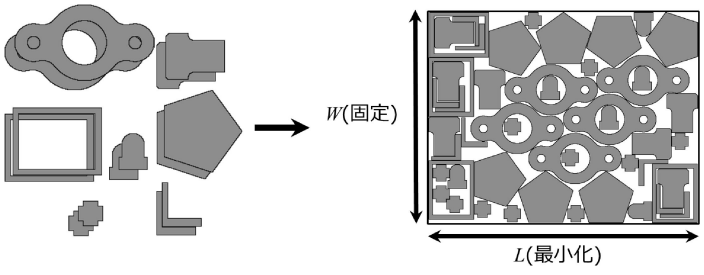

図 5: ストリップパッキング問題の実行可能解の例

4.2 重なり度最小化問題

重なり度最小化問題はストリップパッキング問題の部分問題として扱い, 与えられた長さ $\overline{L}$ の長方形領域 $C(W,\overline{L})$ において実行可能解を発見する問題である. 関数 $f_{ij}(\boldsymbol{r}_i,\boldsymbol{r}_j,o_i,o_j)$ は点 $\boldsymbol{r}_i,\boldsymbol{r}_j$ に配置された図形対 $P_i(o_i),P_j(o_j)$ の重なり度である. 重なり度最小化問題の目的は全ての図形 $P_i(1 \leq i \leq n)$ が幅 W, 長さ $\overline{L}$ の長方形領域 $C(W,\overline{L})$ 内に配置され, 重なり度 $F(\boldsymbol{r},\boldsymbol{o}) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} f_{ij}(\boldsymbol{r}_i,\boldsymbol{r}_j,o_i,o_j)$ を最小化する解を求めることである.

$$\begin{aligned} \text{minimize} \quad & F(\boldsymbol{r},\boldsymbol{o}) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} f_{ij}(\boldsymbol{r}_i,\boldsymbol{r}_j,o_i,o_j) \\ \text{subject to} \quad & (P_i(o_i)\oplus\boldsymbol{r}_i) \subseteq C(W,\overline{L})\ ,\ 1 \leq i \leq n, \\ & o_i \in O_i\ ,\ 1 \leq i \leq n, \\ & \boldsymbol{r}_i \in \mathbb{R}^2\ ,\ 1 \leq i \leq n. \end{aligned} \tag{7}$$

Egebald ら [9] は, 図形対 P_i,P_j が重なっている面積を重なり度 f_{ij} としている. Imamichi ら [12], Umetani ら [16] は, 重なり度 f_{ij} の定義に侵入度を用いている. 点 $\boldsymbol{r}_i,\boldsymbol{r}_j$ に配置された重なりを持つ図形 $P_i(o_i),P_j(o_j)$ の侵入度 $\delta(P_i(o_i)\oplus\boldsymbol{r}_i,P_j(o_j)\oplus\boldsymbol{r}_j)$ は図形 P_j の重なりを解消するための最小移動距離である. 侵入度は式 (8) のように表される.

$$\delta(P_i(o_i)\oplus\boldsymbol{r}_i,P_j(o_j)\oplus\boldsymbol{r}_j) = \min\{\|\boldsymbol{u}\| \mid (P_i(o_i)\oplus\boldsymbol{r}_i)\cap(P_j(o_j)\oplus(\boldsymbol{r}_j+\boldsymbol{u})) = \emptyset, \boldsymbol{u}\in\mathbb{R}^2\}, \tag{8}$$

ここで, $\|\cdot\|$ はユークリッド距離である.

本研究では, 図形対 P_i,P_j の重なりを解消するのに必要な水平方向, 垂直方向の最小移動距離を重なり度 f_{ij} とする. 任意のある一方向 $\boldsymbol{d}=(d_x,d_y)\in\mathbb{R}^2$ における重なりを解消するための最小移動距離は式 (9) のように表される.

$$\rho(P_i(o_i)\oplus\boldsymbol{r}_i,P_j(o_j)\oplus\boldsymbol{r}_j,\boldsymbol{d}) = \min\{|t| \mid (P_i(o_i)\oplus\boldsymbol{r}_i)\cap(P_j(o_j)\oplus(\boldsymbol{r}_j+t\boldsymbol{d})) = \emptyset, t\in\mathbb{R}\}, \tag{9}$$

また, 重なり度 f_{ij} は式 (10) のように表される.

$$f_{ij}(\boldsymbol{r}_i,\boldsymbol{r}_j,o_i,o_j) = \min\{\rho(P_i(o_i)\oplus\boldsymbol{r}_i,P_j(o_j)\oplus\boldsymbol{r}_j,\boldsymbol{d}) \mid \boldsymbol{d}\in\{\boldsymbol{e}_x,\boldsymbol{e}_y\}\} \tag{10}$$

ここで, $\boldsymbol{e}_x=(1,0),\boldsymbol{e}_y=(0,1)$ である.

5 ミンコフスキー差

重なり度最小化問題において重なり度 f_{ij} を効率良く計算するためにミンコフスキー差を用いる. ミンコフスキー差はしばしば No-Fit Polygon と呼ばれる. No-Fit Polygon は Art ら [2] によって導入された図形対の重なりを効率良く判定するためデータ構造である. P_i に対する P_j のミンコフスキー差 $P_i\ominus P_j$ を定義する. 図形 P_i を原点に固定し, 図形 P_j は移動可能とする. このとき, $P_i\ominus P_j$ は図形対が重なりを持つような図形 P_j の配置全体からなる集合であり, 式 (11) のように表される.

$$\mathrm{NFP}(P_i,P_j) = P_i\ominus P_j = \{\boldsymbol{u}-\boldsymbol{w} \mid \boldsymbol{u}\in P_i,\boldsymbol{w}\in P_j\}. \tag{11}$$

図 6 にミンコフスキー差の例を示す. ただし, 図 6 における灰色の図形がミンコフスキー差である.

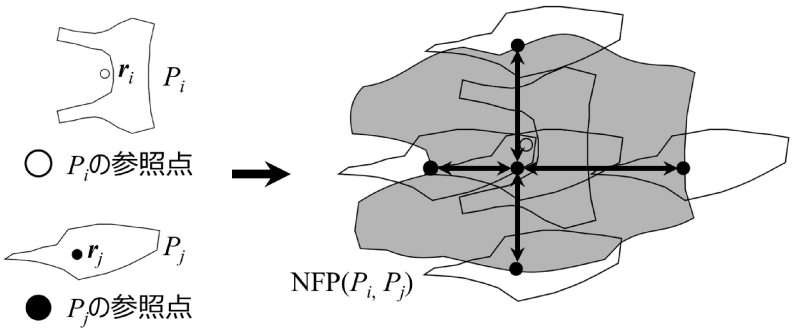

図 6: 図形 P_i, P_j に対するミンコフスキー差

本研究では, 図形 $P_i(1 \leq i \leq n)$ をスキャンラインの集合で表現するため, ミンコフスキー差もスキャンラインの集合で表現される. 図形 $P_i(1 \leq i \leq n)$ は a_i^h 個の水平方向のスキャンラインの集合 $S_i^h = \{S_{i1}^h, S_{i2}^h, \ldots, S_{ia_i^h}^h\}$ により表現される. 各水平方向のスキャンライン $S_{ik}^h(1 \leq k \leq a_i^h)$ は幅 1, 長さ l_{ik}^h である. 水平方向のスキャンラインの集合で表現されたミンコフスキー差 $\mathrm{NFP}^h(P_i, P_j)$ は $a_i^h + a_j^h - 1$ 個のスキャンラインの集合 $S_{ij}^h = \{S_{ij1}^h, S_{ij2}^h, \ldots, S_{ija_i^h+a_j^h-1}^h\}$ により表現される. 同様に, 図形 $P_i(1 \leq i \leq n)$ は a_i^v 個の垂直方向のスキャンラインの集合 $S_i^v = \{S_{i1}^v, S_{i2}^v, \ldots, S_{ia_i^v}^v\}$ により表現される. 各垂直方向のスキャンライン $S_{ik}^v(1 \leq k \leq a_i^v)$ は幅 w_{ik}^v, 長さ 1 である. 垂直方向のスキャンラインの集合で表現されたミンコフスキー差 $\mathrm{NFP}^v(P_i, P_j)$ は $a_i^v + a_j^v - 1$ 個のスキャンラインの集合 $S_{ij}^v = \{S_{ij1}^v, S_{ij2}^v, \ldots, S_{ija_i^v+a_j^v-1}^v\}$ により表現される.

図形 P_i の参照点に対する図形 P_j の参照点の相対座標は $(x_j - x_i, y_j - y_i)$ であり, この参照点を含むミンコフスキー差 $\mathrm{NFP}^h(P_i, P_j)$ のスキャンライン S_{ijk}^h の始点と終点の x 座標を b_{ijk}^l, b_{ijk}^r とする. このとき, 水平方向の重なりを解消するための最小移動距離 $\rho(P_i(o_i) \oplus \boldsymbol{r}_i, P_j(o_j) \oplus \boldsymbol{r}_j, \boldsymbol{e}_x)$ は P_j の参照点から, この参照点を含むスキャンラインの始点 b_{ijk}^l, または終点 b_{ijk}^r までの距離の最小値となる. 水平方向の重なりを解消するための最小移動距離は式 (12) のように表される.

$$\rho(P_i(o_i) \oplus \boldsymbol{r}_i, P_j(o_j) \oplus \boldsymbol{r}_j, \boldsymbol{e}_x) = \min\{((x_j - x_i) - b_{ijk}^l), (b_{ijk}^r - (x_j - x_i))\} \tag{12}$$

同様に, 図形 P_i の参照点に対する図形 P_j の参照点の相対座標は $(x_j - x_i, y_j - y_i)$ であり, この参照点を含むミンコフスキー差 $\mathrm{NFP}^v(P_i, P_j)$ のスキャンライン S_{ijk}^v の始点と終点の y 座標を b_{ijk}^u, b_{jik}^d とする. このとき, 垂直方向の重なりを解消するための最小移動距離 $\rho(P_i(o_i) \oplus \boldsymbol{r}_i, P_j(o_j) \oplus \boldsymbol{r}_j, \boldsymbol{e}_y)$ は P_j の参照点から, この参照点を含むスキャンラインの始点 b_{ijk}^u, または終点 b_{ijk}^d までの距離の最小値となる. 垂直方向の重なりを解消するための最小移動距離は式 (13) のように表される.

$$\rho(P_i(o_i) \oplus \boldsymbol{r}_i, P_j(o_j) \oplus \boldsymbol{r}_j, \boldsymbol{e}_y) = \min\{((y_j - y_i) - b_{ijk}^u), (b_{ijk}^d - (y_j - y_i))\} \tag{13}$$

式 (12), (13) より, 水平方向, 垂直方向の重なりを解消するための最小移動距離を得ることができるため, 式 (10) より重なり度 $f_{ij}(\boldsymbol{r}_i, \boldsymbol{r}_j, o_i, o_j)$ を求めることができる. 図 7 にミンコフスキー差による重なり度の計算の例を示す. 図 7 において, 水平方向の重なりを解消するための最小移動距離は P_j の参照点からこの参照点を含むスキャンラインの左端までの距離であり, 垂直方向の重なりを解消するための最小移動距離は P_j の参照点からこの参照点を含むスキャンラインの下端までの距離であることが分かる.

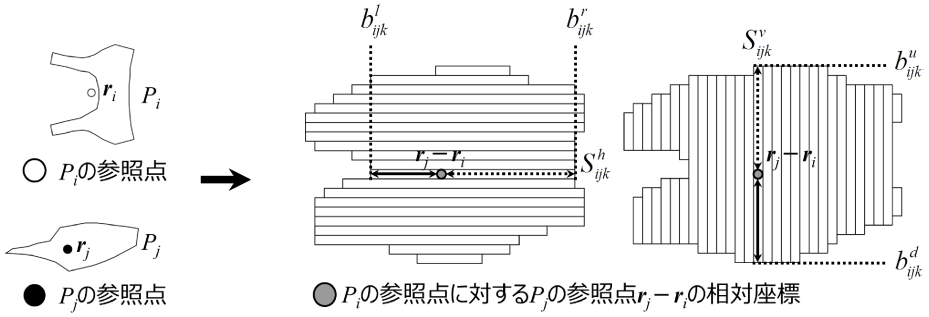

図 7: ミンコフスキー差による図形 P_i, P_j の重なり度 $f_{ij}(\boldsymbol{r}_i, \boldsymbol{r}_j, o_i, o_j)$ の計算

6 アルゴリズムの概要

本研究で用いるストリップパッキング問題に対するアルゴリズムの概要について述べる. 初めに初期解を 7 節で述べる方法で作成する. そして与えられた計算時間 T に達するまで, 下記の手順を繰り返すことで最良解を得る.

初めに, 長方形領域 C の長さ L を縮小または拡張する. 長さ L の縮小または拡張はパラメータ $\Delta^{\mathrm{dec}}, \Delta^{\mathrm{inc}}$ により決定される. 現在の配置が実行可能解であれば, 長方形領域 C の長さを L から $(1-\Delta^{\mathrm{dec}})L$ に縮小し, 長方形領域 C からはみ出した図形 P_i を領域内にランダムに再配置する. 一方で, 現在の配置が実行可能解でなければ, 長方形領域 C の長さを L から $(1+\Delta^{\mathrm{inc}})L$ に拡張する. ここで, 図形対に重なりが存在する場合, 長方形領域 C の長さを L に固定して, 誘導局所探索法を用いて重なり度最小化問題を解き, 重なりを解消する. 誘導局所探索法については 8 節で述べる. 図 8 にアルゴリズムの実行例を示す. ここで, 図 8 における黒色の図形は重なりを持つ図形, 灰色の図形は重なりを持たない図形を示す.

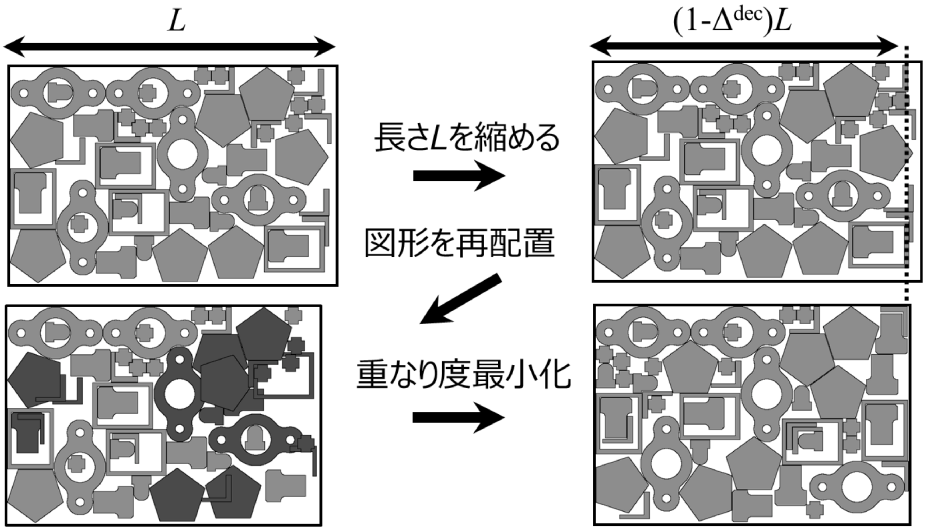

図 8: アルゴリズムの実行例

アルゴリズムの概要

入力: 図形 $P = \{P_1, P_2, \ldots, P_n\}$, 各図形の回転可能な回転角の集合 $O = \{O_1, O_2, \ldots, O_n\}$ と幅 W の長方形領域 C.

出力: 図形 $P_i(1 \leq i \leq n)$ の配置 $\boldsymbol{r} = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n\}$, 回転角 $\boldsymbol{o} = \{o_1, o_2, \ldots, o_n\}$ と長方形領域 C の長さ $L(\boldsymbol{r}, \boldsymbol{o})$.

Step1: 初期解 $(\boldsymbol{r}, \boldsymbol{o})$ を生成し, $L^* \leftarrow L, (\boldsymbol{r}^*, \boldsymbol{o}^*) \leftarrow (\boldsymbol{r}, \boldsymbol{o})$ とする.

Step2: 現在の解 $(\boldsymbol{r}, \boldsymbol{o})$ が実行可能解であれば, 各図形の配置を微調整する. $L < L^*$ を満たす場合, $L^* \leftarrow L, (\boldsymbol{r}^*, \boldsymbol{o}^*) \leftarrow (\boldsymbol{r}, \boldsymbol{o})$ として, $L \leftarrow (1 - \Delta^{\mathrm{dec}})L$ を行う. 一方, 現在の解 $(\boldsymbol{r}, \boldsymbol{o})$ が実行可能解でなければ, $L \leftarrow (1 + \Delta^{\mathrm{inc}})L$ とする. もし, 計算時間が T に達した場合, 長方形領域 C の長さ L^*, 解 $(\boldsymbol{r}^*, \boldsymbol{o}^*)$ を出力して終了する.

Step3: 図形 P_i が長方形領域 C の外にはみ出している場合, 領域 C 内にランダムに再配置する.

Step4: 現在の解 $(\boldsymbol{r}, \boldsymbol{o})$ が実行可能解でなければ, 誘導局所探索法により重なり度を最小化する. そして Step2 に戻る.

7 初期解の構築法

本研究は, Next-Fit Algorithm を用いて各図形を配置し, 得られた各図形の配置の微調整をすることで初期解を構築する. Next-Fit Algorithm は長方形詰込み問題に対する近似解法の 1 つである. 長方形領域をレベルと呼ばれる m 個 の長方形 $V = \{V_1, V_2, \ldots, V_m\}$ に分割し, 各レベルに図形を上詰めに配置する. レベル V_k に図形が配置できない場合, レベル V_{k+1} に図形を配置する. 本研究で Bbounding box を用いて, 各図形を長方形として考えることで, Next-Fit Algorithm により各図形を配置する. 初めに, 図形 $P_i(1 \leq i \leq n)$ の回転角を 0° に固定し, 各図形の長さ l_i を降順に整列する. 各図形 $P_i(1 \leq i \leq n)$ を 1 つずつ, 図形同士が接するように左上に配置する. 図 9 に Next-Fit Algorithm の実行結果の例を示す.

Next-Fit Algorithm

入力: 回転角を 0° に固定された図形 $P = \{P_1, P_2, \ldots, P_n\}$, 各図形の長さ $l_i(1 \leq i \leq n)$ と幅 W の長方形領域 C.

出力: 図形 $P_i(1 \leq i \leq n)$ の初期配置 $\boldsymbol{r} = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n\}$ と長方形領域 C の初期の長さ $L(\boldsymbol{r}, \boldsymbol{o})$.

Step1: $i \leftarrow 1, L \leftarrow 0, l^* \leftarrow 0, w^{\mathrm{sum}} \leftarrow 0$ とする. また, 各図形の長さ l_i を降順に整列した添字の順列 $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ を得る.

Step2: $w^{\mathrm{sum}} + w_{\sigma_i} > W$ を満たす場合, $w^{\mathrm{sum}} \leftarrow 0, L \leftarrow L + l^*, l^* \leftarrow 0$ とする.

Step3: $w^{\mathrm{sum}} + w_{\sigma_i} \leq W$ を満たす場合, 図形 $P_{\sigma_i}(1 \leq i \leq n)$ を $(x_i, y_i) = (L + \lfloor \frac{l_{\sigma_i}}{2} \rfloor, w^{\mathrm{sum}} + \lfloor \frac{w_{\sigma_i}}{2} \rfloor)$ に配置して, $i \leftarrow i + 1, l^* \leftarrow \max\{l^*, l_{\sigma_i}\}$, $w^{\mathrm{sum}} \leftarrow w^{\mathrm{sum}} + w_{\sigma_i}$ とする. もし, $i = n$ ならば終了し, $i \leq n$ ならば Step2 に戻る.

次に, Next-Fit Algorithm で得られた配置に対して, 各図形の配置を微調整する. 配置されている図形 $P_i(1 \leq i \leq n)$ を x_i の昇順に整列して, 図形 $P_i(1 \leq i \leq n)$ を 1 つずつ整列された順に選ぶ. 選択した図形 P_i の左側の各配置に対して, 重なり度 f_{ij} を計算し, 重なり度 $f_{ij} = 0$ を満たす配置の中で, 最も左側の配置に移動させる. 次に, 選択した図形 P_i の上側の各配置に対して, 重なり度 f_{ij} を計算し, 重なり度 $f_{ij} = 0$ を満たす配置の中で, 最も上側の配置に移動させる. この操作を図形 $P_i(1 \leq i \leq n)$ が移動しなくなるまで繰り返し, 重なりが増加しない範囲で長方形領域 C の長さ L を最小化する. 図 10 に各図形の配置の微調整の実行結果の例を示す.

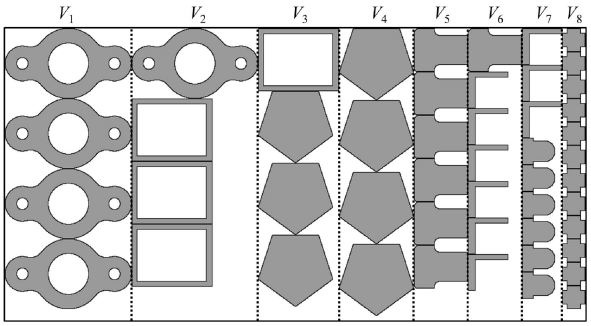

図 9: Next-Fit Algorithm の実行結果の例

各図形の配置の微調整

入力: 回転角を 0° に固定された図形 $P = \{P_1, P_2, \ldots, P_n\}$, 各図形の配置 $\boldsymbol{r} = \{\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n\}$ と幅 W, 長さ L の長方形領域 C.

出力: 図形 $P_i(1 \leq i \leq n)$ の微調整された配置 $r^* = \{r_1^*, r_2^*, \ldots, \quad r_n^*\}$ と長方形領域 C の長さ L.

Step1: 図形 $P_i(1 \leq i \leq n)$ を x_i の昇順に整列した添字の順列 $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ を得る. また, $i \leftarrow 1$ とする.

Step2: $\boldsymbol{r}^*_{\sigma_i} \leftarrow \boldsymbol{r}_{\sigma_i}$ とする.

Step3: 図形 P_{σ_i} が, $(P_{\sigma_i} \oplus (\boldsymbol{r}_{\sigma_i} - \boldsymbol{e}_x)) \cap (P_j \oplus \boldsymbol{r}_j) = \emptyset (1 \leq j \leq n, \sigma_i \neq j)$ かつ $P_{\sigma_i} \oplus (\boldsymbol{r}_{\sigma_i} - \boldsymbol{e}_x) \subseteq C(W, L)$ を満たす場合, $\boldsymbol{r}_{\sigma_i} \leftarrow \boldsymbol{r}_{\sigma_i} - \boldsymbol{e}_x$ とする. もし, $P_{\sigma_i} \oplus (\boldsymbol{r}_{\sigma_i} - \boldsymbol{e}_x) \subseteq C(W, L)$ ならば, Step3 に戻る.

Step4: 図形 P_{σ_i} が, $(P_{\sigma_i} \oplus (\boldsymbol{r}_{\sigma_i} - \boldsymbol{e}_y)) \cap (P_j \oplus \boldsymbol{r}_j) = \emptyset (1 \leq j \leq n, \sigma_i \neq j)$ かつ $P_{\sigma_i} \oplus (\boldsymbol{r}_{\sigma_i} - \boldsymbol{e}_y) \subseteq C(W, L)$ を満たす場合,$\boldsymbol{r}_{\sigma_i} \leftarrow \boldsymbol{r}_{\sigma_i} - \boldsymbol{e}_y$ とする. もし, $P_{\sigma_i} \oplus (\boldsymbol{r}_{\sigma_i} - \boldsymbol{e}_y) \subseteq C(W, L)$ ならば, Step4 に戻る.

Step5: $\boldsymbol{r}^*_{\sigma_i} = \boldsymbol{r}_{\sigma_i}$ を満たす場合, $i \leftarrow i + 1$ として Step2 に戻る. もし, $i > n$ ならば, $\boldsymbol{r}^* = \boldsymbol{r}$ を満たす場合は終了し, 満たさない場合は Step1 に戻る.

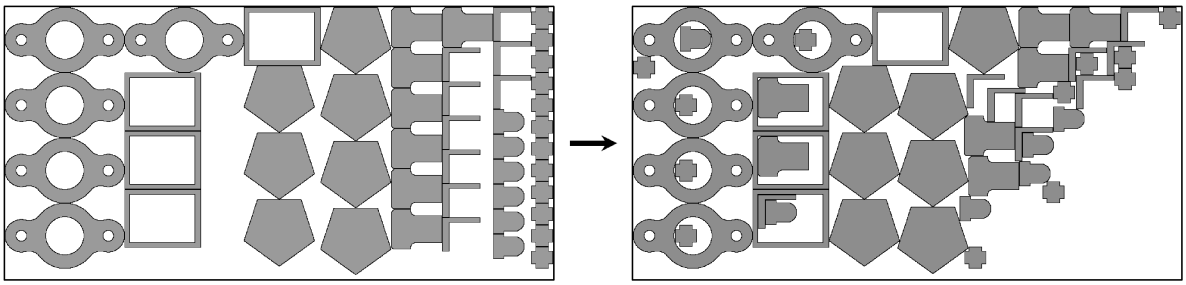

図 10: 各図形の配置の微調整の実行結果の例

8 重なり度最小化問題に対する局所探索法

8.1 局所探索法の概要

局所探索法とは近傍内に改善解が存在する限り, その解に移動する操作を繰り返して, 解を探索する手法である. 本研究で用いる近傍と解の良否の判定条件について説明する. 近傍 $\mathrm{NB}(\boldsymbol{r},\boldsymbol{o})$ は近傍解 $(\boldsymbol{r}',\boldsymbol{o}')$ の集合であり, 近傍解 $(\boldsymbol{r}',\boldsymbol{o}')$ は図形 $P_k(o_k)(1\leq k\leq n)$ に対して近傍探索して得られる解である. 本研究では, 水平方向, 垂直方向を交互に探索し, 図形 $P_k(o'_k)$ の解が改善されなくなるまで近傍探索を繰り返す. 図形 $P_k(o'_k)$ の探索方向は $\boldsymbol{d}\in\{\boldsymbol{e}_x,\boldsymbol{e}_y\}$ であり, 式 (14) のように表される.

$$N_0=\{\boldsymbol{r}_k+t\boldsymbol{d}\mid \boldsymbol{r}_k+t\boldsymbol{d}\subseteq C(W,L)\} \tag{14}$$

そのため, 近傍探索では現在の配置 $\boldsymbol{r}_k$ よりも $F(\boldsymbol{r}'_k,\boldsymbol{o}'_k)$ の値が小さくなる新たな配置 $\boldsymbol{r}'_k=\boldsymbol{r}_k+t\boldsymbol{d}$ を探索する.

本研究では, 本来の評価関数 $F(\boldsymbol{r},\boldsymbol{o})$ の代わりに, ペナルティ重みを導入した式 (15) を用いて解 $(\boldsymbol{r},\boldsymbol{o})$ を評価する.

$$\overline{F}(\boldsymbol{r},\boldsymbol{o})=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}w_{ij}f_{ij}(\boldsymbol{r}_i,\boldsymbol{r}_j,o_i,o_j) \tag{15}$$

ただし, $w_{ij}>0$ のペナルティ重みである. ペナルティ重みの説明および更新については 8.3 節で説明する. また, 図形 $P_k(o'_k)$ の新たな配置 $\boldsymbol{r}'_k$ を探索するときは, 式 (16) が最小化される配置を新たな配置とする.

$$\overline{F}(\boldsymbol{r}'_k,\boldsymbol{o}'_k)=\sum_{1\leq j\leq n\ j\neq k}w_{kj}f_{kj}(\boldsymbol{r}'_k,\boldsymbol{r}_j,o'_k,o_j) \tag{16}$$

図 11 に近傍探索の例を示す.

近傍探索

入力: 図形 $P=\{P_1,P_2,\ldots,P_n\}$, 幅 W, 長さ L の長方形領域 C, 図形 P_k の新たな回転角 o'_k.
出力: 図形 $P_k(o'_k)$ の新たな配置 $\boldsymbol{r}'_k$.
Step1: $\boldsymbol{r}'_k\leftarrow\boldsymbol{r}_k$ とする. 図形 $P_k(o'_k)$ に対して $\overline{F}_k(\boldsymbol{r}'_k+t\boldsymbol{e}_x,o'_k)$ を最小化するような $\boldsymbol{r}''_k=\boldsymbol{r}'_k+t\boldsymbol{e}_x(t\in N_0)$ を探索する. $\overline{F}_k(\boldsymbol{r}''_k,o'_k)<\overline{F}_k(\boldsymbol{r}'_k,o'_k)$ を満たす場合, $\boldsymbol{r}'_k\leftarrow\boldsymbol{r}''_k$ とする. $\boldsymbol{d}\leftarrow\boldsymbol{e}_y$ とする.
Step2: 図形 $P_k(o'_k)$ に対して $\overline{F}_k(\boldsymbol{r}'_k+t\boldsymbol{d},o'_k)$ を最小化するような $\boldsymbol{r}''_k=\boldsymbol{r}'_k+t\boldsymbol{d}(t\in N_0)$ を探索する. もし, $\overline{F}_k(\boldsymbol{r}''_k,o'_k)<\overline{F}_k(\boldsymbol{r}'_k,o'_k)$ を満たす場合, $\boldsymbol{r}'_k\leftarrow\boldsymbol{r}''_k$ とする. $\overline{F}_k(\boldsymbol{r}''_k,o'_k)<\overline{F}_k(\boldsymbol{r}'_k,o'_k)$ を満たす解が存在しなければ, $\boldsymbol{r}'_k$ を出力して終了する.
Step3: $\boldsymbol{d}=\boldsymbol{e}_x$ならば, $\boldsymbol{d}\leftarrow\boldsymbol{e}_y$として, $\boldsymbol{d}=\boldsymbol{e}_y$ならば, $\boldsymbol{d}\leftarrow\boldsymbol{e}_x$ として, Step2 に戻る.

次に局所探索法について説明する. 局所探索法はいくつかの重なりを持つ図形が存在する解 $(\boldsymbol{r},\boldsymbol{o})$ に対して, 近傍 $\mathrm{NB}(\boldsymbol{r},\boldsymbol{o})$ 内の改善解 $(\boldsymbol{r}',\boldsymbol{o}')$ に移動させることを繰り返すことで解を改善する. 近傍 $\mathrm{NB}(\boldsymbol{r},\boldsymbol{o})$ 内の改善解 $(\boldsymbol{r}',\boldsymbol{o}')$ を探索する順番はランダムに選択する. $\overline{F}(\boldsymbol{r}',\boldsymbol{o}')<\overline{F}(\boldsymbol{r},\boldsymbol{o})$ を満たす改善解 $(\boldsymbol{r}',\boldsymbol{o}')\in\mathrm{NB}(\boldsymbol{r},\boldsymbol{o})$ を探索し, 改善解が存在すれば移動する. もし, 重なりが存在しないか, 近傍 $\mathrm{NB}(\boldsymbol{r},\boldsymbol{o})$ に改善解が存在しなければ, 解 $(\boldsymbol{r},\boldsymbol{o})$ と評価関数 F が最も最小化された解 $(\boldsymbol{r}^*,\boldsymbol{o}^*)$ を出力する.

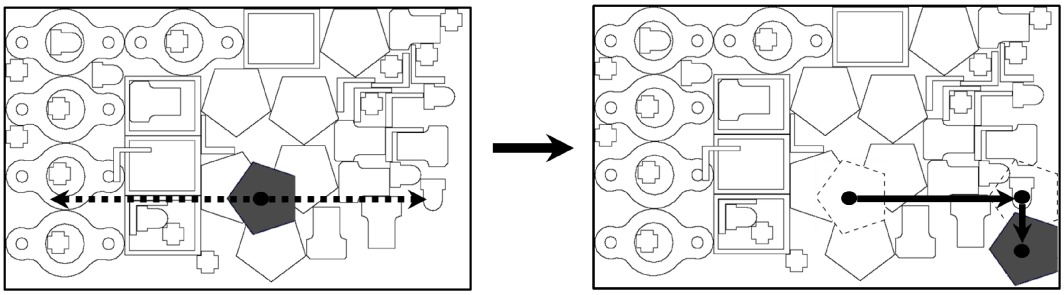

図 11: 近傍探索の例

探索する各図形 $P_k(o'_k)$ $(o'_k \in O_k)$ に対して, 部分近傍 $\mathrm{NB}_k(\boldsymbol{r},\boldsymbol{o})$ $(1 \leq k \leq n)$ を定義する. 近傍 $\mathrm{NB}(\boldsymbol{r},\boldsymbol{o})$ は図形 P_k $(1 \leq k \leq n)$ の部分近傍 $\mathrm{NB}_k(\boldsymbol{r},\boldsymbol{o})$ $(1 \leq k \leq n)$ の集合である. 初めに, 重なりを持つ全ての図形の部分近傍 $\mathrm{NB}_k(\boldsymbol{r},\boldsymbol{o})$ $(1 \leq k \leq n)$ を作成する. ランダムな順番で部分近傍 $\mathrm{NB}_k(\boldsymbol{r},\boldsymbol{o})$ $(1 \leq k \leq n)$ を探索する. 部分近傍 $\mathrm{NB}_k(\boldsymbol{r},\boldsymbol{o})$ $(1 \leq k \leq n)$ に $\overline{F}(\boldsymbol{r}',\boldsymbol{o}') < \overline{F}(\boldsymbol{r},\boldsymbol{o})$ を満たすような解が存在しない場合, 探索を終了する. もし, 部分近傍 $\mathrm{NB}_k(\boldsymbol{r},\boldsymbol{o})$ $(1 \leq k \leq n)$ 内で $\overline{F}(\boldsymbol{r}',\boldsymbol{o}') < \overline{F}(\boldsymbol{r},\boldsymbol{o})$ を満たすような近傍解 $(\boldsymbol{r},\boldsymbol{o})$ が存在するならば, 移動前か移動後の図形 P_k と重なる図形 P_j $(j \neq k)$ の部分近傍 $\mathrm{NB}_j(\boldsymbol{r},\boldsymbol{o})$ を探索する部分近傍に加える. アルゴリズムを以下に示す. ただし, Q は探索する部分近傍 $\mathrm{NB}_k(\boldsymbol{r},\boldsymbol{o})$ の添字 k $(1 \leq k \leq n)$ の集合である. また, $(\overline{\boldsymbol{r}},\overline{\boldsymbol{o}})$ は評価関数 $\overline{F}$ によって評価された局所最適解である.

局所探索法

入力: 図形 $P = \{P_1, P_2, \ldots, P_n\}$, 各図形の回転可能な回転角の集合 $O = \{O_1, O_2, \ldots, O_n\}$, 幅 W, 長さ L の長方形領域 C と解 $(\boldsymbol{r},\boldsymbol{o})$.

出力: 評価関数 F が最も最小化された解 $(\boldsymbol{r}^*,\boldsymbol{o}^*)$, 評価関数 $\overline{F}$ が最も最小化された解 $(\overline{\boldsymbol{r}},\overline{\boldsymbol{o}})$.

Step1: $(\overline{\boldsymbol{r}},\overline{\boldsymbol{o}}) \leftarrow (\boldsymbol{r},\boldsymbol{o})$, $(\boldsymbol{r}^*,\boldsymbol{o}^*) \leftarrow (\boldsymbol{r},\boldsymbol{o})$ として, $Q \leftarrow \{1, 2, \ldots, n\}$ とする

Step2: $Q = \emptyset$ を満たす場合, $(\boldsymbol{r}^*,\boldsymbol{o}^*)$, $(\overline{\boldsymbol{r}},\overline{\boldsymbol{o}})$ を出力して終了する. $Q \neq \emptyset$ を満たす場合, ランダムに $k \in Q$ を選択し, $O \leftarrow O_k$ とする.

Step3: ランダムに $o'_k \in O$ を選択して, 図形 $P_k(o'_k)$ に対して近傍探索を適用して, 近傍解 $(\boldsymbol{r}',\boldsymbol{o}')$ を得る. $F(\boldsymbol{r}',\boldsymbol{o}') < F(\boldsymbol{r}^*,\boldsymbol{o}^*)$ を満たす場合, $(\boldsymbol{r}^*,\boldsymbol{o}^*) \leftarrow (\boldsymbol{r}',\boldsymbol{o}')$ とする. もし, $\overline{F}(\boldsymbol{r}',\boldsymbol{o}') < \overline{F}(\overline{\boldsymbol{r}},\overline{\boldsymbol{o}})$ ならば, $(\overline{\boldsymbol{r}},\overline{\boldsymbol{o}}) \leftarrow (\boldsymbol{r}',\boldsymbol{o}')$ として Step4 へ進む. $\overline{F}(\boldsymbol{r}',\boldsymbol{o}') < \overline{F}(\overline{\boldsymbol{r}},\overline{\boldsymbol{o}})$ でないなら, $O \leftarrow O \setminus \{o'_k\}$ とする. もし, $O = \emptyset$ ならば, $Q \leftarrow Q \setminus \{k\}$ として, Step2 に戻る. $O \neq \emptyset$ ならば, Step3 に戻る.

Step4: $F(\boldsymbol{r}^*,\boldsymbol{o}^*) = 0$ を満たす場合, $(\boldsymbol{r}^*,\boldsymbol{o}^*)$, $(\overline{\boldsymbol{r}},\overline{\boldsymbol{o}})$ を出力して, 終了する. $F(\boldsymbol{r}^*,\boldsymbol{o}^*) \geq 0$ を満たす場合, 近傍探索を行う前, 近傍探索を適用した後の図形 P_k と重なる全ての図形 P_j に対して, $Q \leftarrow Q \cup \{j\}$ として, Step2 に戻る.

8.2 近傍探索の高速化

本研究では, 近傍内の任意の配置で重なる可能性がない図形との重なり度の計算を省略することで近傍探索を高速化する.

$$I_{kj} = \{t \mid \boldsymbol{r}_k + t\boldsymbol{d} - \boldsymbol{r}_j \in \mathrm{NFP}(P_j(o_j), P_k(o_k)\} \tag{17}$$

$I_{kj} = \emptyset$ を満たす場合, 2 つの図形 $P_j(o_j), P_k(o_k)$ は重なる可能性がない. ここで, 図形 P_i の x 座標の左端と右端, y 座標の上端と下端を式 (18), (19), (20), (21) に示す.

$$x_i^{\min} = \min\{x \mid (x, y) \in P_i(o_i) \oplus \boldsymbol{r}_i\} \tag{18}$$

$$x_i^{\max} = \max\{x \mid (x, y) \in P_i(o_i) \oplus \boldsymbol{r}_i\} \tag{19}$$

$$y_i^{\min} = \min\{y \mid (x, y) \in P_i(o_i) \oplus \boldsymbol{r}_i\} \tag{20}$$

$$y_i^{\max} = \max\{y \mid (x, y) \in P_i(o_i) \oplus \boldsymbol{r}_i\} \tag{21}$$

水平方向の近傍探索の場合, $(y_k^{\min}, y_k^{\max}) \cap (y_j^{\min}, y_j^{\max}) = \emptyset$ を満たすならば, $I_{kj} = \emptyset$ を満たす. 同様に, 垂直方向の近傍探索の場合, $(x_k^{\min}, x_k^{\max}) \cap (x_j^{\min}, x_j^{\max}) = \emptyset$ を満たすならば, $I_{kj} = \emptyset$ を満たす. このため, 重なる可能性がない図形の集合を得ることができ, 重なり度の計算を省略することができる. 図 12 に近傍探索の高速化の例を示す. ただし, 図 12 における黒色の図形は探索する図形, 灰色の図形は重なる可能性がある図形, 白色の図形は重なる可能性がない図形を示す.

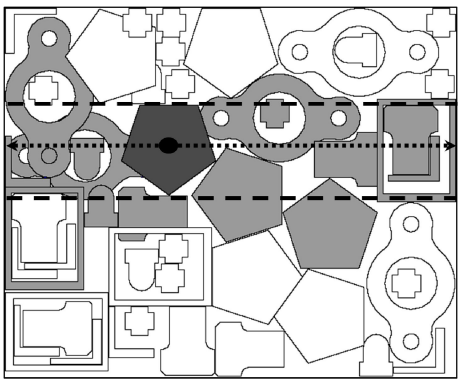

図 12: 近傍探索の高速化の例

8.3 誘導局所探索法

単純な局所探索法では, しばしば精度の低い局所最適解に陥る. そのため, ペナルティ重みの適応的な調整を用いた誘導局所探索法を用いる. 初めにペナルティ重み w_{ij} を 1.0 に初期化する. 局所最適解に陥ったとき, 以下の式 (22) に従ってペナルティ重み w_{ij} を更新することで, 局所最適解に陥った際にも, 探索を再開することができる. 点 $\boldsymbol{r}_i, \boldsymbol{r}_j$ に配置された図形 $P_i(o_i)$ と $P_j(o_j)$ が重なりを持つ場合, ペナルティ重みの更新は式 (22) に従う.

$$w_{ij} \leftarrow w_{ij} + \frac{f_{ij}(\boldsymbol{r}_i, \boldsymbol{r}_j, o_i, o_j)}{\max\limits_{1 \le k < l \le n} f_{kl}(\boldsymbol{r}_k, \boldsymbol{r}_l, o_k, o_l)}. \tag{22}$$

誘導局所探索法

入力: 図形 $P = \{P_1, P_2, \ldots, P_n\}$, 各図形の回転可能な回転角の集合 $O = \{O_1, O_2, \ldots, O_n\}$, 幅 W, 長さ L の長方形領域 C と解 $(\boldsymbol{r}, \boldsymbol{o})$.

出力: 長方形領域 C の長さ $L(\boldsymbol{r}^*, \boldsymbol{o}^*)$ と図形 $P_i(1 \leq i \leq n)$ の配置 $\boldsymbol{r}^* = \{\boldsymbol{r}_1^*, \boldsymbol{r}_2^*, \ldots, \boldsymbol{r}_n^*\}$, 回転角 $\boldsymbol{o}^* = \{o_1^*, o_2^*, \ldots, o_n^*\}$.

Step1: $max\_iter \leftarrow 0, w_{ij} \leftarrow 1(1 \leq i, j \leq n), (\boldsymbol{r}^*, \boldsymbol{o}^*) \leftarrow (\boldsymbol{r}, \boldsymbol{o})$ とする.

Step2: 重なりを持つ図形 P_i に対して局所探索法を適用して, 重なりが最小となる配置に図形を移動させることで, 解 $(\boldsymbol{r}, \boldsymbol{o})$ を更新する.

Step3: $F(\boldsymbol{r}, \boldsymbol{o}) < F(\boldsymbol{r}^*, \boldsymbol{o}^*)$ を満たす場合, $(\boldsymbol{r}^*, \boldsymbol{o}^*) \leftarrow (\boldsymbol{r}, \boldsymbol{o})$ として, $iter \leftarrow 0$ とする. もし, $F(\boldsymbol{r}, \boldsymbol{o}) = 0$ ならば, 解 $(\boldsymbol{r}^*, \boldsymbol{o}^*)$ を出力して終了する.

Step4: $iter \leftarrow iter + 1$ とする. もし, $iter \geq max\_iter$ ならば, 式 (22) を用いて $w_{ij}(1 \leq i, j \leq n)$ を更新して, Step2 に戻る.

9 数値実験

本研究では, 円弧や穴を含むベンチマーク問題例 [8] と多角形のベンチマーク問題例 [8, 16] を用いた. これらのベンチマーク問題例の概要を表 1, 2 に示す. ベンチマーク問題例のデータ形式はベクタ図形であるため, 長方形領域 C の幅 $W = 512$ ピクセルとして, 各図形をラスタ形式に変換した.

表 1: 円弧や穴を含むベンチマーク問題例

問題名	図形の種類	図形数	回転角の増分 (°)
Progfiles1	8	32	90
Progfiles2	7	50	90
Progfiles3	6	46	45
Progfiles4	7	54	90
Progfiles5	5	50	15
Progfiles6	9	69	90
Progfiles7	9	9	90
Progfiles8	9	18	90
Progfiles9	16	57	90
Progfiles10	13	91	0

本研究では提案手法の実験結果を Burke ら [8], Umetani ら [16] の結果と比較する. Burke ら [8] はベクタ図形を用いて, 円弧や穴を含むベンチマーク問題例, 多角形のベンチマーク問題例を扱っている. Umetani ら [16] はベクタ図形を用いて, 多角形のベンチマーク問題例を扱っている. また, Umetani ら [16] の手法は従来の多角形のベンチマーク問題例の最良解をいくつか更新している.

本研究では, ペナルティ重みの更新回数 $max\_iter = 200$, 領域の幅を調整するパラメータ $l^{\mathrm{dec}} = 0.02$, $l^{\mathrm{inc}} = 0.005$ とする. 各ベンチマーク問題例に対して提案手法を 10 回適用し, 得られた解の中で最も充填率が高い結果を解とした. ただし, 充填率は $\sum_{i=1}^{n}(P_i\text{の面積})/WL$ としており, 各実行で乱数の種を変更している. また, 現在の提案手法の計算効率を改善した場合に, どの程度向上できるか確かめるために, 多角形のベンチマーク問題例に対して 24 時間の実験を 1 回行った.

表 1, 2 のベンチマーク問題例に対して得られた結果と Bukre ら [8] の手法 BLF, Umetani ら [16] の手法 FITS の結果を表 3, 4 に示す. ただし, 各結果は表 5, 6 に示した計算機環境, 計算時間で得

表 2: 円弧や穴を含むベンチマーク問題例

問題名	図形の種類	図形数	回転角の増分 (°)
Albano	8	24	180
Dagli	10	30	180
Dighe1	16	16	0
Dighe2	10	10	0
Fu	12	12	90
Jakobs1	25	25	90
Jakobs2	25	25	90
Mao	9	20	90
Marques	8	24	90
Shapes0	4	43	0
Shapes1	4	43	180
Shirts	8	99	180
Swim	10	48	180
Trousers	17	64	180

られた結果である. 提案手法と Umetani ら [16] は 1200 秒の制限時間を設けている. Burke ら [8] は制限時間を設けていないが, 他の基準を終了条件として用いており, 表 5, 6 は表 3, 4 の計算結果を得るまでにかかった時間を示している.

表 3: 円弧や穴を含むベンチマーク問題例に対する実験結果

問題名	BLF	提案手法	
	Best(%)	Best(%)	Avg.(%)
Progfiles1	82.5	<u>84.2</u>	82.3
Progfiles2	73.8	<u>76.2</u>	75.0
Progfiles3	70.8	<u>72.3</u>	70.8
Progfiles4	<u>86.8</u>	84.0	82.7
Progfiles5	75.9	<u>81.2</u>	80.0
Progfiles6	72.1	<u>78.8</u>	77.5
Progfiles7	73.3	<u>95.8</u>	95.5
Progfiles8	78.7	<u>82.9</u>	81.9
Progfiles9	52.9	<u>57.5</u>	56.4
Profiles10	73.2	<u>78.0</u>	66.1

表 4: 多角形のベンチマーク問題例に対する実験結果

問題名	BLF	FITS		提案手法 (1200 秒)		提案手法 (24 時間)
	Best(%)	Best(%)	Avg.(%)	Best(%)	Avg.(%)	Best(%)
Albano	86.0	90.0	87.3	87.5	86.5	87.7
Dagli	82.2	86.9	85.7	85.2	84.4	86.0
Dighe1	82.1	99.9	99.8	96.9	86.1	97.6
Dighe2	84.3	100.0	100.0	97.3	97.2	97.3
Fu	89.2	91.2	90.3	89.5	87.7	90.7
Jakobs1	82.6	89.1	88.7	84.5	83.1	86.8
Jakobs2	75.1	80.4	80.3	78.2	77.1	79.1
Mao	78.7	85.0	82.8	83.9	82.8	84.6
Marques	86.5	89.8	88.8	89.0	88.4	89.7
Shapes0	65.6	66.8	66.2	64.9	64.1	65.3
Shapes1	71.5	73.7	72.6	71.6	69.7	71.8
Shirts	82.8	87.2	86.1	84.2	83.3	85.2
Swim	67.2	74.9	73.0	72.6	71.1	73.4
Trousers	86.9	89.5	88.8	86.8	85.6	87.5

表 5: 円弧や穴を含むベンチマーク問題例に対する計算機環境と計算時間 (秒)

問題名	BLF	提案手法
	Pentium4	Core i7
	2.0GHz	3.1GHz
	10runs	10runs
	Best	Time Limit
Progfiles1	15	1200
Progfiles2	295	1200
Progfiles3	283	1200
Progfiles4	256	1200
Progfiles5	300	1200
Progfiles6	171	1200
Progfiles7	211	1200
Progfiles8	279	1200
Progfiles9	98	1200
Profiles10	247	1200

表 6: 多角形のベンチマーク問題例に対する計算機環境と計算時間 (秒)

問題名	BLF Pentium4 2.0GHz 10runs Best	FITS Core i7 3.1GHz 10runs Time Limit	提案手法 Core i7 3.1GHz 10runs Time Limit
Albano	299	1200	1200
Dagli	252	1200	1200
Dighe1	3	1200	1200
Dighe2	148	1200	1200
Fu	139	1200	1200
Jakobs1	29	1200	1200
Jakobs2	51	1200	1200
Mao	152	1200	1200
Marques	21	1200	1200
Shapes0	274	1200	1200
Shapes1	239	1200	1200
Shirts	194	1200	1200
Swim	141	1200	1200
Trousers	253	1200	1200

提案手法は Burke ら [8] の結果を数多くのベンチマーク問題例で上回った. この結果から, 提案手法は円弧や穴を含むベンチマーク問題例においては十分な性能を示すことが分かった. 一方で, Umetani ら [16] の結果を全てのベンチマーク問題例で下回った. この理由について考察する. 局所探索法の呼び出し回数と長方形領域 C の幅 W の関係について図 13 に示す. 図 13 より, 長方形領域 C の幅 W が大きくなる程, 局所探索法の呼び出し回数が減少していることが分かる. これは長方形領域 C の幅 W が大きくなるにつれて近傍探索で重なり度を評価する配置の数が増加するためだと考えられる.

次に, 局所探索法の呼び出し回数が十分か確かめるために, Umetani ら [16] の局所探索の呼び出し回数の比較を表 7 に示す. 7 より, 提案手法の局所探索法の呼び出し回数が Umetani ら [16] の手法と比較すると非常に少ないことが分かり, そのため充填率が低い可能性があることが考えられる.

次に, 現在の局所探索法の計算効率を改善した場合に, 充填率がどの程度向上できるか確かめる. 表 4 より 24 時間の実行をした場合, 充填率は向上したが, Umetani ら [16] の結果には及ばないことが分かった. この結果から, 現在の提案手法の高速化のみでは不十分であることが分かった. 最後に, 提案手法で得られた各ベンチマーク問題例に対する解を図 14, 15 に示す.

10 結論

本研究では, ラスタ図形の詰込み問題に対する局所探索法を提案した. ラスタ図形は図形対の重なり判定をドット同士の重なりの有無で判定するため, 曲線など複雑な形状の図形の重なり判定においても, ドット同士の重なりの有無で判定できる. また, 重なりを解消するのに必要な移動距離を効率良く計算するためにスキャンライン形式を使用した. 本研究はストリップパッキング問題の

表 7: 多角形のベンチマーク問題例に対する局所探索法の呼び出し回数

問題名	FITS Core i7 3.1GHz 10runs Time Limit	提案手法 Core i7 3.1GHz 10runs Time Limit
Albano	593056	39754
Dagli	401265	31696
Dighe1	1035355	96501
Dighe2	1946249	224961
Fu	1172243	81895
Jakobs1	336613	33541
Jakobs2	289989	29396
Mao	824967	76821
Marques	248231	59821
Shapes0	298206	39173
Shapes1	463018	41578
Shirts	94094	9413
Swim	76822	19022
Trousers	233491	28613

部分問題である重なり度最小化問題を繰り返し解くことでストリップパッキング問題の解を求めた. 重なり度最小化問題では, 長方形領域の幅と長さを固定することで, 図形対の重なり度の総和を最小化した. 重なり度は重なりを解消するのに必要な水平方向, 垂直方向の最小移動距離としている. また, ミンコフスキー差を用いることで効率的に重なり度を計算することができた. 重なり度最小化問題の解法は局所探索法を使用した. 任意の図形を選択し, 水平方向, 垂直方向に交互に移動させることで, 重なり度を最小化した. 単純な局所探索法では, しばしば精度の低い局所最適解に陥るため, ペナルティ重みの適応的な更新を用いた誘導局所探索法を使用した. 数値実験の結果より, 円弧や穴を含むベンチマーク問題例に対して十分な性能を示すことができた.

参考文献

[1] A. Albano and G. Sapuppo, Optimal allocation of two-dimensional irregular shapes using heuristic search methods, *IEEE Transactions on Systems, Man and Cybernetics*, **10** (1980), 242-248.

[2] R. C Art, An approach to the two dimensional irregular cutting stock problem, *Technical Report 36.Y08, IBM Data Processing Division*, **171** (1966).

[3] A .R. Babu and N. R. Babu, A generic approach for nesting of 2-D parts in 2-D sheets using genetic and heuristic algorithms, *Computer-Aided Design*, **33** (2001), 879-891.

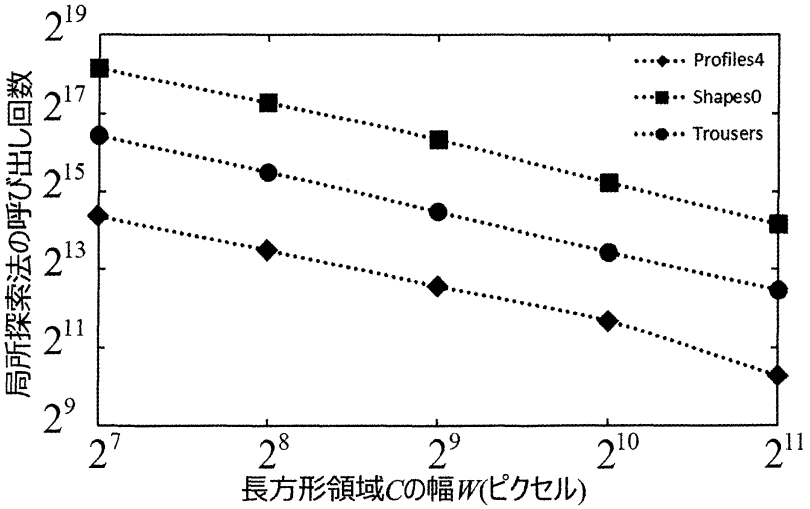

図 13: 局所探索法の呼び出し回数と長方形領域 C の幅 W の関係

[4] J. A. Bennell and K. A. Dowsland, Hybridising tabu search with optimisation techniques for irregular stock cutting, *management Science*, **47** (2001), 1160-1172.

[5] J. A. Bennell and J. F. Oliveira, The geometry of nesting problems: A tutorial, *European Journal of Operational Research*, **184** (2008), 397-415.

[6] J. Blazewicz, P. Hawryluk and R. Walkowiak, Using a tabu search approach for solving the two-dimensional irregular cutting problem, *Annals of Operations Research*, **41** (1993), 313-325.

[7] E. K. Burke, R. S. R. Hellier, G. Kendall and G. Whitwell, A New bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem, *Operations Research*, **54** (2006), 587-601.

[8] E. K. Burke, R. S. R. Hellier, G. Kendall and G. Whitwell, Irregular packing using the line and arc no-fit polygon, *Operations Research*, **171** (2010), 948-970.

[9] J. Egebald, B. K. Nielsen, and A. Odgaard, Fast neighborhood search for two-and three-dimensional nesting problems, *European Journal of Operational Research*, **183** (2007), 1249-1266.

[10] A. M. Gomes and J. F. Oliveira, A 2-exchange heuristic for nesting problmes, *European Journal of Operational Research*, **141** (2002), 359-370.

[11] A. M. Gomes and J. F. Oliveira, Solving irregular strip packing problems by hybridising simulated annealing and linear programming, *European Journal of Operational Research*, **171** (2006), 811-829.

[12] T. Imasmichi, M. Yagiura and H. Nagamochi An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem, *Discreate Optimization*, **6** (2009), 345-361.

[13] H. Okano, A scanline-based algorithm for the 2D free-form bin packing problem, *Journal of Operations Research Society of Japan*, **45** (2002), 145-161.

[14] J. F. Oliveira and J. S. Ferreira, Algorithms for nesting problems, *Lecture Notes in Economics and Mathematical Systems*, **396** (1993), 256-273.

[15] S. A. Segenreich and L. M. Braga Optimal nesting of general plane figures: A monte carlo heuristical approach , *Computers & Graphics*, **10** (1986), 229-237.

[16] S. Umetani, M. Yagiura, S. Imahori, T. Imamichi, K. Nonobe and T. Ibaraki, Solving the irregular strip packing problem via guided local search for overlap minimization, *International Transactions in Operational Research*, **16** (2009), 661-683.

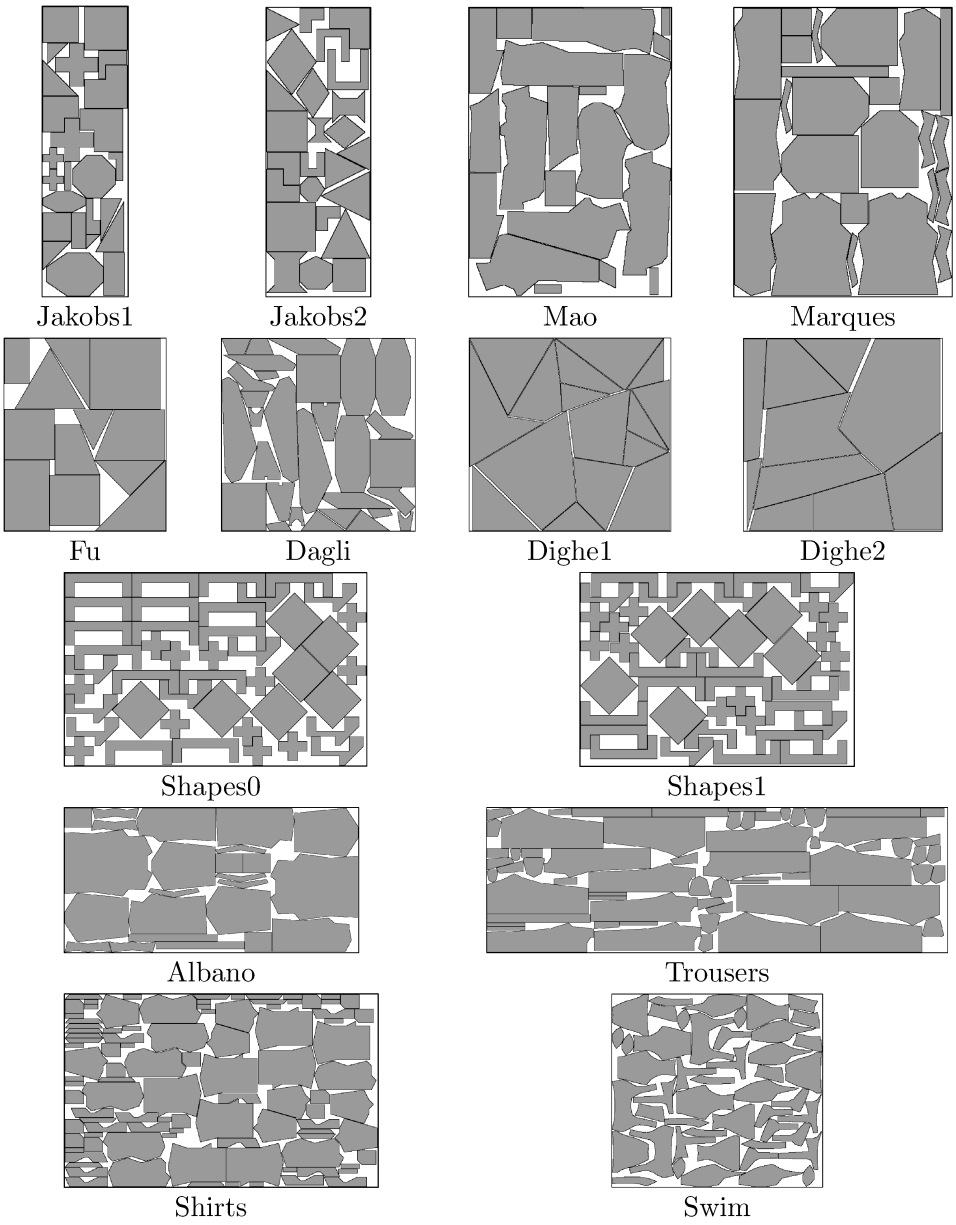

図 14: 提案手法により得られた多角形のベンチマーク問題例に対する解

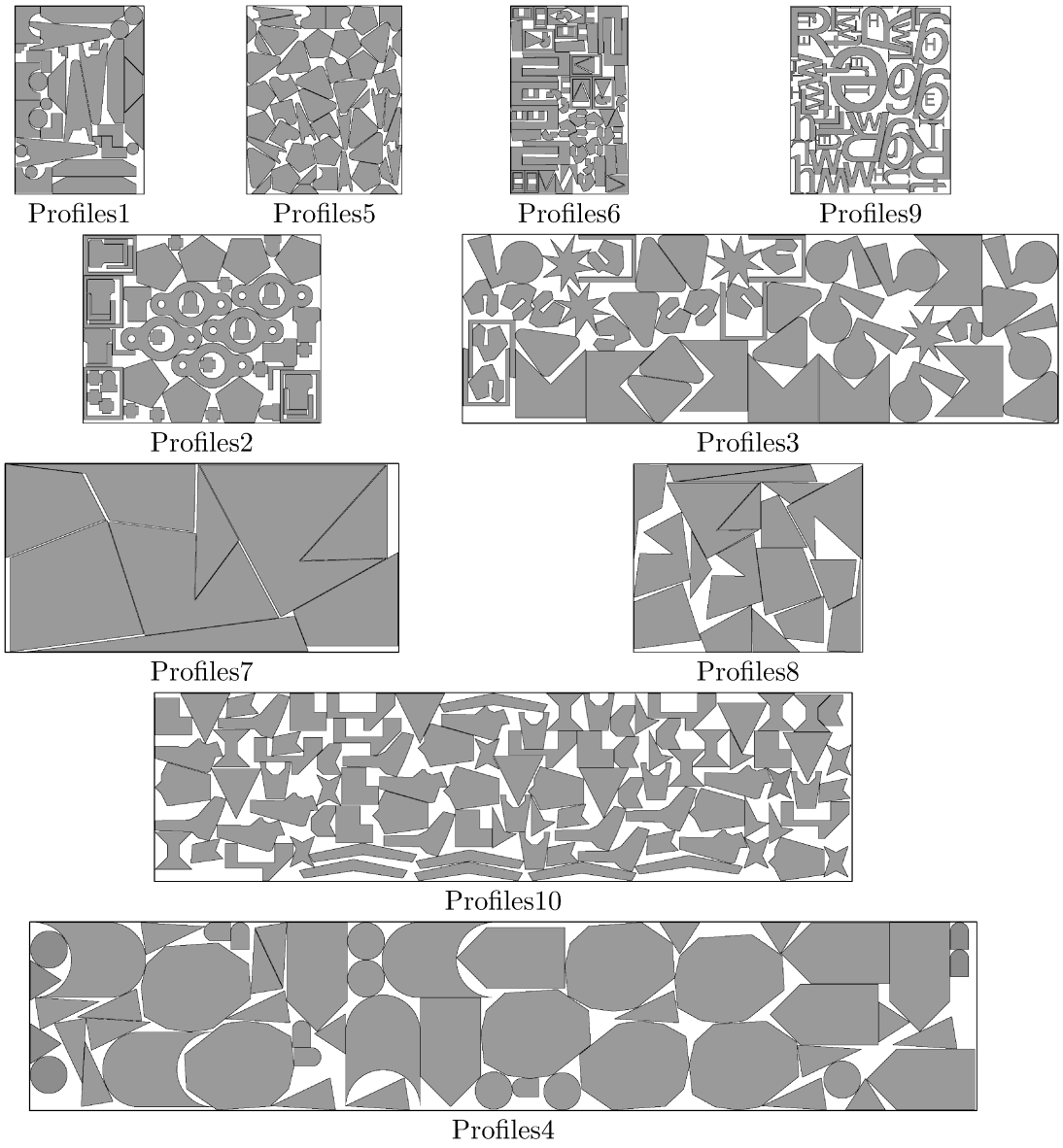

図 15: 提案手法により得られた円弧や穴を含むベンチマーク問題例に対する解