パズルゲーム「タングラム」解法の基本アルゴリズム

Basic Algorithms for Solving the Puzzle Game "Tangram"

大槻 正伸・中野 良樹*・新井 広**
福島工業高等専門学校電気工学科
*秋田大学教育文化学部
**高知工科大学
Masanobu Ohtsuki, Yoshiki Nakano,Hiroshi Arai
National Institute of Technology, Fukushima College, Department of Electrical Engineering
*Akita University, Faculty of Education and Human Studies
**Kochi University of Technology
(2015年9月8日受理)

"Tangram" is a puzzle game, using seven pieces of 5 triangles, a square, and a parallelogram, which construct an original big square with no intersection (i.e. these pieces are parts of a big square). A problem is expressed by a figure, shown in silhouette, which can be constructed with all these seven pieces with no intersection. The problem solver, given these seven pieces, and a problem silhouette figure, have to construct the problem figure with all these seven pieces.

In this paper, we have developed basic algorithms for solving this puzzle instead of human solver, and constructed an algorithm which can solve "2-pieces Tangram", so called simplified Tangram , and it is left to the future research to develop an efficient algorithm for solving the original 7-pieces Tangram.

Key words: Tangram, Computational Geometry

1. はじめに

「タングラム」とは次のようなパズルである。

まずFig.1(左)のように、大きな正方形を分割してできる7個の「ピース」と「問題図形」（例えばFig.1(右)）ーピースで構成すべき影絵の図形ーが与えられる。パズルを解く者は7個のピースを全て用いて、重ねることなく問題図形を構成する（Fig.2）[4) 5) 6) 7)]。

このパズルは、小学校などにおいて算数教育の教材としても使われ、またタングラムを人間が解く場合の問題解決方法について心理学的な研究も行われている[4) 5)]。

本論文では、図形を扱うことが苦手なコンピュータに、タングラム問題を解かせるにはどうしたらよいか、そのアルゴリズムを設計するための研究を行う。

計算機科学（Computer Science）においては、この種の図形問題は、その一分野である「計算幾何学（Computational Geometry）」で扱うべき性質のものであるが[2) 3)]、計算幾何学の中では「多角形領域の基本図形への分解」等の関連しそうなテーマの議論はあるが、直接タングラムの解法に結び付くものはほとんどない。そこで本研究は、タングラムを解くアルゴリズムの基礎研究をすることにより計算幾何学を一歩進めようとするものである。

Fig.1 The basic 7-pieces of Tangram(left) and an example of a problem "Wildcat (reduced scale 0.7)"(right)

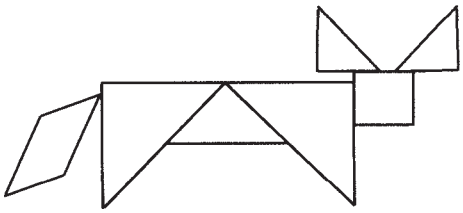

Fig.2 The Solution to the Problem"Wildcat" in Fig.1

「タングラム」という場合、ピースはFig.1のように分割した7個で行うものを指すが、この種のパズルでは類似したものもあり、ピースの形は別のものも考えられている。また元の正方形の分割の仕方もより複雑にもできるし、分割してピースをつくる元の図形を正方形でないものとするパズルもある。　そこで一般的に、

(1)ピースは閉じた有限の図形であり、ピースのエッジ（辺）は全て直線とする。

(2)ピース数 p、ピースの形に関するデータ数（情報の量　例えば各ピースの端点数合計）が c_p である。

(3)指定されたシルエット図形（その情報量 c_s）を、与えられたピースで、ピースの重なりなしで構成する。

という図形組み合わせ問題を考える。

これを「一般的なタングラム問題」あるいは「p（ピース）タングラム問題」などとよぶことにする。

一般的なタングラム問題では、P や c_p 、c_s が大きくなった場合、「解候補が与えられると、それが正しい解であるかどうかは比較的簡単に（i.e.　p や c_p 、c_s に関する多項式時間で）分かるが、初めから解を求めようとすると難しそう（どんなに効率的なアルゴリズムでも p や c_p 、c_s の指数関数時間かかりそう）である」ことが直観されることから、NP 完全問題 [1] に属するものと予想される。一般的タングラム問題が実際に NP 完全問題かどうかについてはまた別の議論が必要であり、それはそれで大きなテーマであるが本論文の主題ではない。

本論文では、ピース数 p=7、端点数合計 c_p =23、c_s =3～30 程度の、NP 完全問題と予想されるタングラム問題を、コンピュータで解くための、データ表現方法、図形を扱う基本的アルゴリズムを考える。以下、2.では一般的タングラム問題をコンピュータで扱うためのデータ表現法を提案し、それで表現された図形を扱う基本的なアルゴリズムを準備し、基本アルゴリズムを用いて一般的タングラムを解くアルゴリズム設計を目指す。

ここで提案するアルゴリズムで、2 ピースのタングラム問題は解けたが 3 ピース以上のタングラム問題は解けなかった。3 ピース以上のタングラム問題、あるいはふつうの 7 ピースタングラム問題を解くアルゴリズムの開発は今後の課題として残されている。

2. 図形の表現方法と基本アルゴリズム

2.1 ピースと問題図形の表現方法

図形（以下、「図形」という場合、ピースや問題のシルエット図形を意味するものとする）は、コンピュータに与えるために、何らかの数値データとして表現しなければならない。　ここでは、以後の説明のために、単純な次の2ピースタングラム問題の例（Fig.3）で説明する。

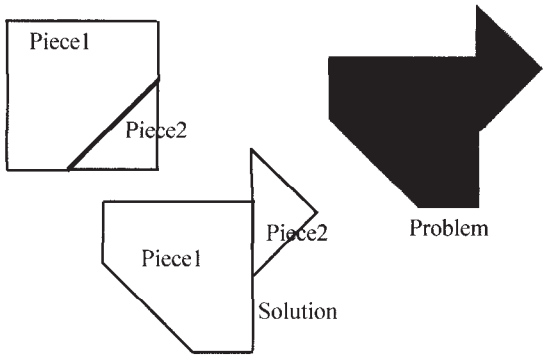

Fig.3 An Example of 2-pieces Tangram Problem

Fig.3は　ピース数 p=2、ピース 1 のデータ数c_{p1}=5　ピース2のデータ数c_{p2}=3、ピースのデータ数c_p=c_{p1}+c_{p2}=8、問題シルエットのデータ数=c_s=8の例である。

図形を定義するために与えるデータはいろいろ考えられるが、ここでは自然に「図形内部を常に左に見るように頂点を順番に訪れるものとし、その順番に頂点座標をデータとして与える」こととする。

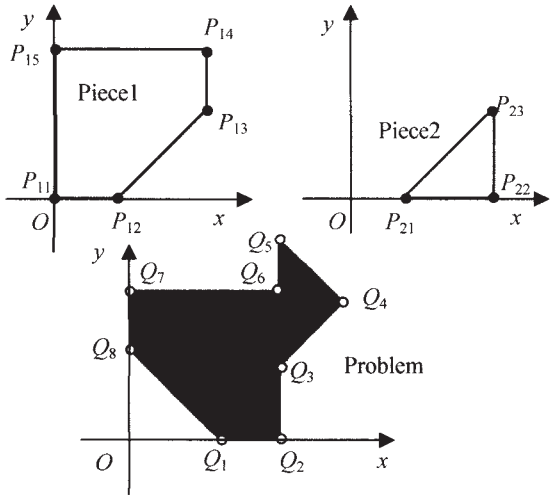

Fig.4 Expression of the 2-pieces Tangram Problem

すなわち、例えば　Fig.3のピース 1（Piece1）であれば、Fig.4に示す各頂点を用いて{P_{11}, P_{12}, P_{13}, P_{14}, P_{15}}でもよいし{P_{13}, P_{14}, P_{15}, P_{11}, P_{12}}でもよい。この表記は集合の記号と同じであるが、この図形の表現においては、点の順序が重要であることに注意する。また、{P_{12}, P_{11}, P_{15}, P_{14}, P_{13}}や、{P_{11}, P_{12}, P_{15}, P_{14}, P_{13}}などはここでいう「正しいピース 1（Piece1）の図形表現」にはなっていないことに注意する。

さらに、この図形表現においては、どの {P_{1k}, $P_{1(k+1)}$, $P_{1(k+2)}$ } 3点も 1 つの直線上にないものとする。

すなわち、例えば $\{P_{1k}(0,0),\ P_{1(k+1)}\ (1,0), P_{1(k+2)}\ (2,0), P_{1(k+3)}\ (2,1)\}$ の場合は、$\{P_{1k}(0,0),\ P_{1(k+1)}\ (2,0), P_{1(k+2)}\ (2,1)\}$ と、無駄をなくして図形を表現しておくものとする。

実際には、例えばピース１ $\{P_{11},\ P_{12},\ P_{13},\ P_{14},\ P_{15}\}$ の場合は $\{P_{11}(0.0, 0.0),\ P_{12}(0.4, 0.0),\ P_{13}(1.0, 0.6),\ P_{14}(1.0, 1.0),\ P_{15}(0.0, 1.0)\}$ のように、2次元の点の座標としてデータが与えられる。そうすると、図形の全ての点を同じだけ平行移動したり、同じだけ回転させたり、線対称に裏返したりしても、同じ図形を表すことになる。すなわち、例えばピース１をベクトルv=(0.1,0.2)だけ平行移動した $\{P_{11}'(0.1, 0.2),\ P_{12}'(0.5, 0.2),\ P_{13}'(1.1, 0.8),\ P_{14}'(1.1, 1.2),\ P_{15}'(0.1, 1.2)\}$ 等もピース１を表す正しい表現である。

なお、ピース１のデータ数c_{p1}＝5という場合、「平面上の5点により表現される図形」ということであり、実際には、2× c_{p1}＝10個の実数データで表現される。この2倍のずれは、計算量のオーダーの議論には影響しないからデータ数はc_{p1}としておく。

また、本論文のアルゴリズムでは、ピースの線対称の変換（裏返し操作）については考えず、ピースを平行移動、回転のみで問題シルエットが構成できるようなもののみを扱うこととする。これは、問題の本質を明らかにするための簡単化という意味と、人間がタングラムを解く際の心理的な実験[4)5)]との比較を行うことを目指すという意味がある。文献4), 5)では、心理学的考察を効率よく行うために、線対称変換を用いない問題を扱っている。なお、コンピュータによる解法において、線対称変換なしの解法プログラムが構築できれば、線対称変換を必要とする問題への対応するプログラムは容易に構築できる。

さて、Fig.3のピース１（Piece1）、ピース2（Piece2）と問題(Problem)の図形もデータ表現すると一例として

Piece1＝$\{P_{11}(0.0, 0.0),\ P_{12}(0.4, 0.0),\ P_{13}(1.0, 0.6),\ P_{14}(1.0, 1.0),\ P_{15}(0.0, 1.0)\}$

Piece2＝$\{P_{21}(0.4, 0.0),\ P_{22}(1.0, 0.0),\ P_{23}(1.0, 0.6)\}$

Problem＝$\{Q_1(0.6, 0.0),\ Q_2(1.0, 0.0),\ Q_3(1.0, 0.5),\ Q_4(1.424264\ldots, 1.0),\ Q_5(1.0, 1.348528\ldots),\ Q_6(1.0, 1.0),\ Q_7(0.0,1.0),\ Q_8(0.0, 0.6)\}$ と表現される。

実際には、コンピュータには「$1+0.3\sqrt{2}$」ではなく「1.424264…」のように小数点をもつ数値データで与える必要がある。

なお、本論文の以下のアルゴリズムで「点P_0」などが出てくるが、点P_0は点P_n, P_{n+1}はP_1, P_{n+2}はP_2等と同一視する。i.e.番号はローテーションして考えることとする。

2.2 図形を扱う基本アルゴリズム

ここで、いくつか図形を扱う基本的なアルゴリズム（A1～A4）を以下のように構成する。

（基本アルゴリズム　A1）［正規化］

ここで、正規化について定義する。

本論文でいう「図形 $\{P_1,\ P_2, \cdots,\ P_n\}$ の点 P_k による正規化」とは、図形と図形の１つの頂点 P_k を与え、P_k を（平行移動で）原点に、その後（回転により）辺 $P_k P_{k+1}$ を x 軸の正の方向に一致させるように図形を合同変換することとする（Fig.5）。なお、この正規化後の表現は、$\{P_1', P_2' \ldots\} = \{P_1$ を平行移動と回転した点 P_1', P_2 を平行移動と回転した点 $P_2', \ldots\}$ であり、点の順序は変えないものとする。

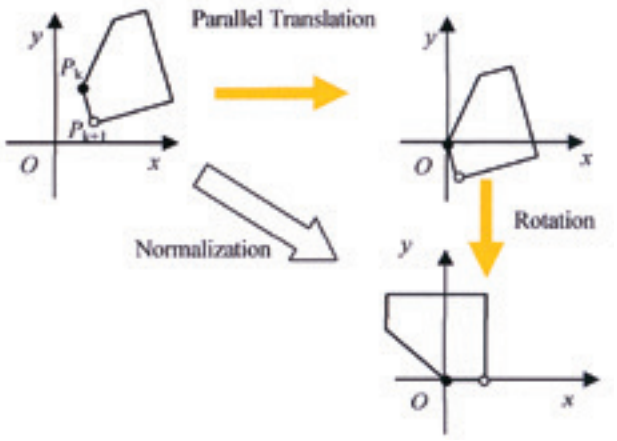

Fig.5 Normalization of a figure by P_k

【アルゴリズムA1】

正規化アルゴリズムは、図形の頂点数をnとするとO(n)時間で実行可能なものが簡単に構成できる（詳しいアルゴリズムは明らかであるからここでは省略する）。

（基本アルゴリズム　A2）［2つの図形の合同判定］

【アルゴリズムA2】

入力（F_1＝ $\{Q_1,Q_2,\ldots, Q_{n1}\}$ ， F_2＝ $\{P_1, P_2,\ldots, P_{n2}\}$ ）

begin

cflag:=0;

if $n1$＝$n2$ **then**

begin

F_{work1}:=(F_1を点Q_1で正規化した図形) ;

for i:=1 **to** n2 **do**

begin

F_2を$\{P_1', P_2', \cdots, P_{n2}'\} = \{P_i, P_{i+1}, \ldots\}$と表現しなおす($P_i$ を表現の開始とする) ;

F_{work2}:=(F_2を点P_1'で正規化した図形) ;

```
            pflag:=1;
            for j:=1 to n2 do
              if F_work1の第j点≠F_work2の第j点
                then    pflag:=0;
            if pflag=1 then cflag:=1
          end;
    end;
  if cflag=1 then F1とF2は合同と判断
             else F1とF2は合同でないと判断
end ;
```

この合同判定アルゴリズムの時間計算量は、$O(\max(n1,n2)^2)$　である。

また、実際にプログラム設計するには、「２点$P_1(x_1,y_1)$と$P_2(x_2,y_2)$が等しいかどうか」の判定には、「$x_1=x_2$かつ$y_1=y_2$」ではなく、「$|x_1-x_2|\le\varepsilon$かつ$|y_1-y_2|\le\varepsilon$」等、許容誤差εを定めておき「ε以下の違いであれば等しいとする」など、コンピュータのデータ表現の有限性を考慮する必要がある。点の回転操作等を行うと、無理数、無限小数の扱いが必要となり、コンピュータ内では、例えば、real（float）宣言した変数で点の座標を表現すると、10^{-15}程度の誤差がどうしても避けられないためである。

（基本アルゴリズム　A3-1）［点∈図形の判断、i.e. 平面上の任意の点Bと、図形$F=\{P_1, P_2, \cdots, P_n\}$を与え、点$B$が図形$F$の内部の点かどうか判断するアルゴリズム（ただし境界線の点、i.e.ある辺P_kP_{k+1}上にある点は内部の点とする）］

このアルゴリズムは、「線分がある図形に完全に含まれるか？」「ある図形Aが図形Bに含まれるか？」等の判断をするアルゴリズムの基礎となる重要なものである。その基本となる考え方は次のとおりである。

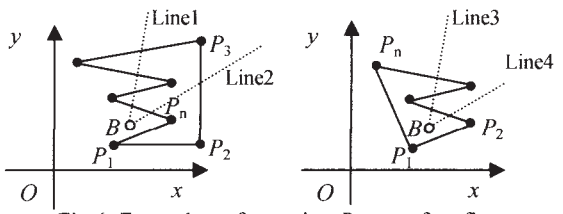

Fig.6 Examples of a point B out of a figure F(left), and B in F(right)

点Bから、任意の方向に半直線を引き、図形の各辺P_1P_2, P_2P_3, …, $P_{n-1}P_n$, P_nP_1と交差する回数を調べる。この半直線上を無限遠点から点Bに向かって点が動く場合、一回交差するごとに、「図形内部に入る」、「図形外部に出る」を繰り返すから、その交差回数が偶数であれば点Bは図形外部の点、奇数であれば内部の点と判断される（Fig.6）。

【アルゴリズムA3-1】

入力　（点$B(x_b,y_b)$、図形F $\{P_1, P_2,\ldots, P_n\}$）

```
begin
  onflag:=0;   count:=0;
  どの線分 PkPk+1（k=1,2,..,n）の傾きとも異なる傾きm
  （m>0）を乱数を用いて選び、点Bから傾きmの半直線
  を引く(これをLine1とする); {Line1上に、ある線分PkPk+1
  が乗っていると処理が複雑になるためこれを避ける}
  for k:=1 to n do
    begin
      if （線分PkPk+1とLine1に共通点がある） and
       (その共通点はPk ,Pk+1ではない) then
            count:=count+1 ;
      if （点Bが線分Pk-1Pk上にある) then   onflag:=1
    end ;
  if (count mod 2=0) and (onflag=0)
     then 点Bは図形Fの外部と判断
     else 点Bは図形Fの内部と判断
end.
```

このアルゴリズムの計算量は$O(n)$である。

アルゴリズムA3-1で、Line1と線分P_kP_{k+1}との交点がちょうど端点（P_kまたはP_{k+1}）の場合、処理は複雑になる。今回は、「Line1と端点が交点」が発生した場合、「乱数でのm(Line1の傾き)の選択やり直し」としている。

（基本アルゴリズム　A3-2）［線分BC全体が図形Fの内部に存在するかどうかの判断をするアルゴリズム］

ここでは単純に、線分BC上の点を、点Bから微小幅hごとに図形Fの内部の点かどうかを（アルゴリズムA3-1で）チェックする（Fig.7）。チェックした点全てが図形の内部であれば線分BCはFの内部にあると判断する。

今回は　$h=10^{-3}$としてプログラムを設計した。ただし、図形Fが凸図形でない場合「解像度」hによっては正しく判断されない場合もある（Fig.7右）が、hを十分小さくすることで精度を上げ、実用的にはほぼ問題なく判定することができる。

正確な「線分$BC\subset$図形F の判定」アルゴリズムの構成は非常に難しい問題である。効率よくこの判定をするアルゴリズム開発は今後の課題として残されている。

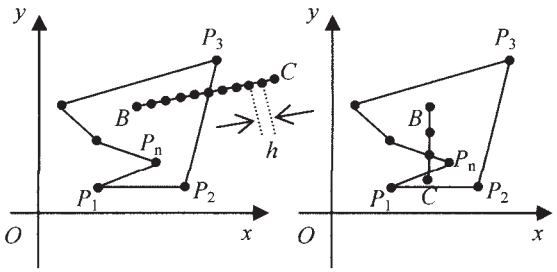

Fig.7 An algorithm for judging whether a segment BC is in a figure F or not

【アルゴリズムA3-2】

入力（点$B(x_b,y_b)$, 点$C(x_c,y_c)$, 図形 $F\{P_1, P_2,..., P_n\}$ ）

begin

inflag:=1;

for i:=0 **to** $\overline{BC}/h$ **do** {＊ $\overline{BC}$ は線分BCの長さ＊}

begin

$D(x_d,y_d):=B(x_b,y_b)+i*h\ (\ C(x_c,y_c)-\ B(x_b,y_b))$;

if （$D \notin F$ ） **then** inflag:=0 ;

{＊ $D \notin F$かどうかの判断はA3-1による ＊}

end ;

if inflag=0 **then** 外部と判断 **else** 内部と判断

end.

このアルゴリズムの計算量は$O(\frac{1}{h}n)$となる。

（基本アルゴリズム A3-3）

図形$F_1=\{Q_1,Q_2,..., Q_{n1}\}$が図形$F_2=\{P_1, P_2,..., P_{n2}\}$の内部にあるかどうかの判定アルゴリズム（Fig.8）

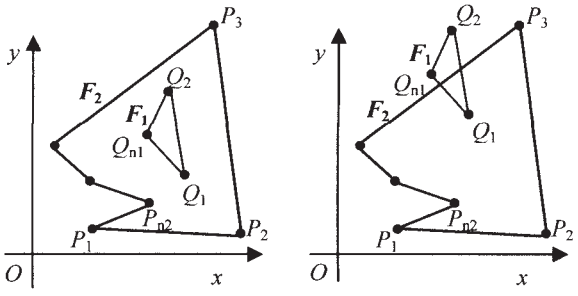

Fig.8 Examples of $F_1 \subset F_2$ (left) and $F_1 \not\subset F_2$ (right)

【アルゴリズムA3-3】

入力（図形 $F_1\{Q_1, Q_2,..., Q_{n1}\}$ 図形 $F_2\{P_1, P_2,..., P_{n2}\}$ ）

begin

inflag:=1;

for i:=1 **to** n_1 **do**

begin

if （線分$Q_i Q_{i+1} \not\subset F$ ） **then** inflag:=0 ;

{＊$Q_i Q_{i+1} \not\subset F$の判断はA3-2による ＊}

end ;

if inflag=0 **then** $F_1 \subset F_2$と判断 **else** $F_1 \not\subset F_2$と判断

end.

このアルゴリズムの計算量は$O(\frac{1}{h}n_1n_2)$である。

（基本アルゴリズム A4）（図形の減算）

図形 $F_1\{Q_1, Q_2,..., Q_{n1}\}$、図形 $F_2\{P_1, P_2,..., P_{n2}\}$ が与えられ、(C1)$F_1 \subset F_2$, (C2)$P_1=Q_1$, (C3)線分$Q_1Q_2 \subset$線分P_1P_2 のとき、F_2-F_1を求めるアルゴリズム（Fig.9）

ここでは、一般的な2つの図形でなく、上記(C1)～(C3)の強い条件を与えて、この条件が満たされるときに、正しく「図形の減算――（F_2-F_1）の図形を正しい表現で求めること」ができるアルゴリズムを設計した。

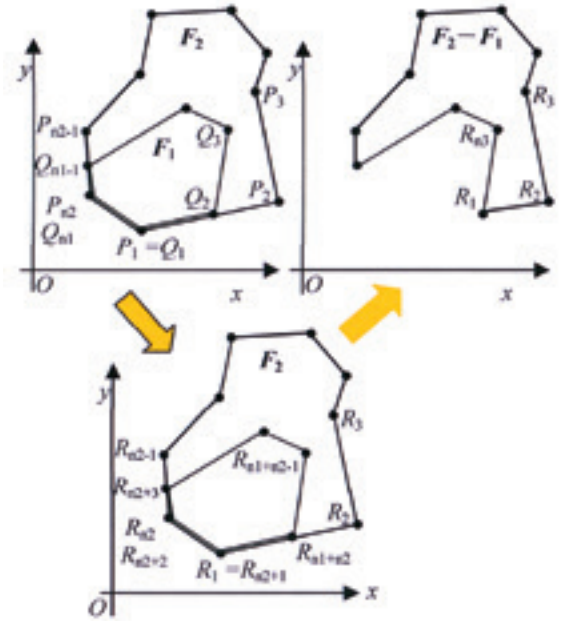

Fig.9 Subtraction of two figures（F_2-F_1）

基本的な考え方は「F_2の各頂点を正順で回り、次にF_1の各頂点を逆順に回りF_3をつくる（Fig.9（下））。そして、『どの連続する3点も１つの直線上にあってはいけない』から、その無駄を全て省く」というものである。

【アルゴリズムA4】

入力（$F_1\{Q_1, Q_2,..., Q_{n1}\}$, $F_2\{P_1, P_2,..., P_{n2}\}$ ）

begin

$F_3=\{R_1, R_2, R_3,...,R_{n1+n2-1}, R_{n1+n2},\}$

$:=\{P_1, P_2,..., P_{n2}, Q_1, Q_{n1},..., Q_2,\}$;

while(ある$R_kR_{k+1}R_{k+2}$が１つの直線上にある)**do**

begin

無駄を省く ;

{＊例えば R_{n1+n2}, R_1, R_2は1つの直線上にあるから、

F_3={ R_{n1+n2} ,R_2, R_3,…,$R_{n1+n2-1}$ }とする等＊}

end

end.

この図形減算アルゴリズムの計算量はO($(n_1+n_2)^2$)

2.3　2-タングラムの解法アルゴリズム

前節2.2の基本アルゴリズムをもとに、次の2-タングラムを解くアルゴリズムを設計した。

【アルゴリズム　2-Tangram】

入力（$F_{piece(1)}\{P_{11}, P_{12}, \ldots, P_{1n1}\}$, $F_{piece(2)}\{P_{21}, P_{22}, \ldots, P_{2n2}\}$
$F_{problem}\{Q_1, Q_2, \ldots, Q_{n3}\}$）

//以下{A-X}はアルゴリズムA-Xを呼び出すことを示す

begin

$F_{PNormalize}$:= $F_{problem}$を点Q_1で正規化した図形；{A-1}

Solvedflag:=0;

for i:=1 to 2 **do**

for j:=1 **to** n_i **do**

begin

F_{Pij}:=$F_{piece(i)}$を点P_{ij}で正規化した図形；{A-1}

if　$F_{Pij} \subset F_{PNormalize}$　**then**{A3-3}

begin

F_{work} := $F_{PNormalize} - F_{Pij}$　；{A-4}

if $F_{piece(3-i)} \equiv F_{work}$ **then** {A-2}

begin　Solvedflag:=1;

解の情報（F_{pij},F_{work}）を記憶

end

end

end；

if Solvedflag=1 **then**

begin

write('Solved')；　解の表示

end

else

write(' Not Solved')

end.

このアルゴリズムの時間計算量は$O(\frac{1}{h}\max(n_1,n_2,n_3)^3)$

このアルゴリズムにより、Delphiを用いてプログラムをパソコンに実装し、実行した結果の例をFig10に示す。

問題図形はFig4をπ/20だけ回転した図形とした。

上段には、問題と得られた解が示され、下段には、正規化された問題図形と２つのピースが表示されるようにしてある。その他の2-ピースタングラム問題もこのプログラムで解けることが確認されている。

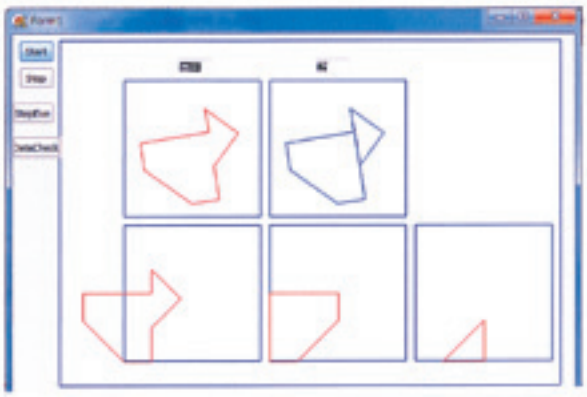

Fig.10　Result of the Program Executed

3. まとめ

一般のタングラム問題をコンピュータで解くための図形表現法、基本アルゴリズムを設計し、今回は、2ピースタングラムまで解けるアルゴリズムを開発した。

今後の課題としては次のことがあげられる。

・本来の7-ピースタングラムが解けるアルゴリズム開発

・アルゴリズムA-3と同等のよりよいアルゴリズム開発

・例えばGA（遺伝的アルゴリズム）等を用いた高速で、高い確率で解が求められるアルゴリズムの開発

・一般的な見地での数学モデル（一般的タングラム問題の公理系）の構築

参 考 文 献

1) A.V. Aho, J.E.Hopcroft, J.D.Ullman, The Design and Analysis of Computer Algorithms, pp364-404, Addison－Wesley Publishing Company,1974

2) 浅野 哲夫, 計算幾何学, 朝倉書店, 1990

3) M.ドバーグ, O.チョン, M.ファンクリベルド, M.オーバマーズ著（浅野 哲夫訳）, コンピュータ・ジオメトリ, 近代科学社, 2010

4) 中野 良樹, 児玉 佳一, 数理パズル「タングラム」の洞察的問題解決における解決を予測する要因の探索, 秋田大学教育文化学部研究紀要　教育科学第 69 集, pp121-131, 2014

5) 中野 良樹, 数理パズル「タングラム」における洞察的問題解決, 秋田大学教育文化学部研究紀要　教育科学第 64 集, pp65-72, 2009

6)タングラムに関するホームページ URL (1)
http://hp.vector.co.jp/authors/VA010128/math/tangram/tframe01.html

7) タングラムに関するホームページ URL (2)
http://ja.wikipedia.org/wiki/%E3%82%BF%E3%83%B%E3%82%B0%E3%83%A9%E3%83%A0