信州大学工学部

学士論文

正方パターンの敷き詰めによる二次元コードの探索と分類に関する基礎的考察

指導教員　西新 幹彦 准教授

学科　電子情報システム工学科
学籍番号　18T2109B
氏名　中川 七海

2022 年 2 月 27 日

目次

1　序論

1.1　研究の背景

二次元シンボルは，バーコードシンボル体系(barcode symgology)の一つであり，光学的反射率の高い部分と低い部分の組み合わせで情報を表示し，機械的に読み取り可能にした情報媒体である[1]．二次元シンボルは，一次元シンボルや RFID などと共に生産現場や広告などを始めとし，近年では乗車券や決済サービスなどの多岐にわたる場面で使用されており，これらを総称してデータキャリアと呼ぶ．図 1.1 にデータキャリアの例を示す．データキャリアは紙面や液晶などのあらゆる場所で表示・読み取りができ利用しやすいことから，情報伝達の手段として広く普及している．中でも，二次元シンボルは横方向のみに情報を持つバーコードなどの一次元シンボルに対し，水平方向と垂直方向に情報を持つシンボルであるため，小さい面積に印刷可能，かつ，集積できる情報量が多いことが特徴である．近年では二次元シンボルのマトリックス内に位置検出のためのファインダーパターン（切り出しシンボル）と呼ばれる特徴的な図形が配置され，シンボルの位置や大きさ，傾きを検出でき，360°全方向から読み取ることができるものも実用化されている．一方，ファインダーパターンをマトリックス内に配置すると，印刷面積が縮小されるため集積できる情報量が減少してしまうという欠点がある．

本研究では，ファインダーパターンをマトリックス内に配置せず，$n \times n$マトリックスを上下左右に大量に敷き詰めることによって位置検出を不要とする二次元シンボルの表示方式の提案に向けて，その基本的性質を調査した．本手法は，ファインダーパターンがないために簡易なマトリックスにすることができ，かつ，印刷面積を最大限に集積したい情報のために使用できることから情報密度の増加に期待できる．

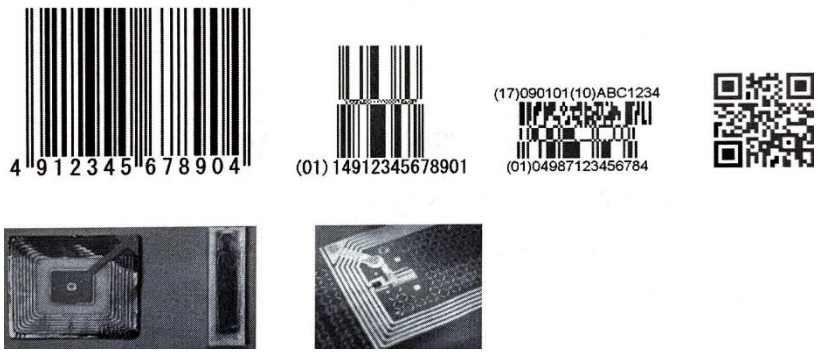

図 1.1 データキャリアの例[1]

1.2　研究の目的

本研究では，二次元シンボルのマトリックス内にファインダーパターンを配置することによって位置検出を可能にするという方法に対し，$n \times n$マトリックスを紙面上に上下左右に大量に敷き詰めることによって位置検出を不要とした方法の提案を目指し，集積できる情報量の探索と，シンボルの性質の調査を目的としている．本研究では，C 言語を用いて集積できる情報量の計算を行った．しかし，パターンが$2^{n \times n}$個と，nの増加に伴い膨大な数になり$n = 6$以降，第一案として考えたアルゴリズムでは計算が長時間に及んだため，$n = 1,2,3,4,5$のマトリックスの結果から以降の結果を推測するという方法で調査を試みた．

1.3　本論文の構成

本論文は全 4 章で構成されている．第 1 章では，本研究の背景と目的を述べる．第 2 章では，従来のバーコードシンボルの概要と二次元シンボルの例について述べる．第 3 章では，本研究で検討する正方形$n \times n$パターン敷き詰めによる二次元コードの概要と，集積可能な情報量について，$n = 1$~5の調査結果と$n = 6$以降のマトリックスについての推測を考察する．それを踏まえ本シンボルにおける符号化レートについて考察する．またシンボルを特徴ごとに分類し，規則性を見定める．第 4 章では，本研究を総括し結論を述べる．

2　従来のバーコードシンボル

2.1　緒言

バーコードシンボル体系には一次元シンボルと二次元シンボルがある．一次元シンボルと二次元シンボルにはそれぞれ数十種類以上のシンボルが実用化されており，さらに各研究機関が新たなバーコードシンボルの提案を行っている．

一次元シンボルは日本では一般的にバーコードと呼ばれ，高い信頼性が特徴である．バーをどこでも横切ってスキャンさえすれば読み取りが可能であるので，図 2.1(a)に示すようにバーのどこかに傷や汚れがあっても読み取りが可能である[1]．この高い信頼性から，小売業界や物流業界，医療業界などで利用されている．

二次元シンボルはシンボルの中に集積できる情報量が多いことが特徴である．図 2.1(b)に示すように最大情報量が 1KB 以上のものもあり，英数字なら約 2000 字以上，数字なら約 3000 桁以上を一つのシンボルにコード化することができる[2][3]．この情報密度の高さから，入出荷明細書や現品ラベル，電子データ交換，小物商品識別などで利用されている．

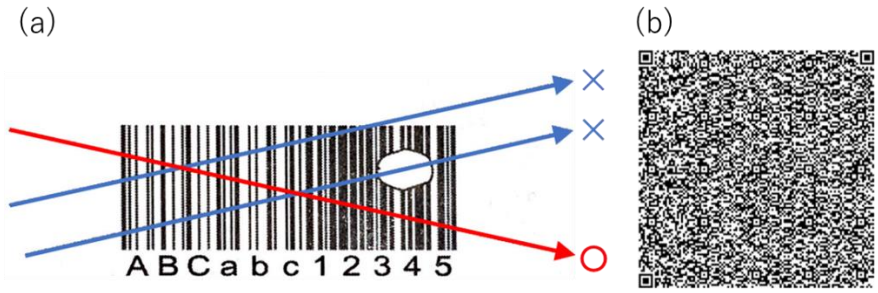

図 2.1 (a) バーコードの走査ライン
(b)約 1KB の指紋・DNA 情報が書き込まれた QR Code[3]

このように，バーコードシンボル体系は安価なメディアであるにも関わらず，高い信頼性や高い情報密度を持つことから，社会で広く利用されており，情報社会のインフラとして重要な役割を果たしている．

本研究では二次元シンボルを考察する．そのため，本章では二次元シンボルの概要と広く普及している 4 つの二次元シンボルの特徴と利点・欠点について述べ，一次元シンボルについては割愛する．

2.2　二次元シンボルの概要

1970 年，IBM 社(International Business Machines Corporation)は世界で初めての一次元シンボルである UPC シンボルを開発した．この UPC シンボルは，格納できる情報が 13 桁の数字のみであったが，情報をコンピュータへ自動入力することが可能となった点で画期的な発明であった．世界で急激に情報化が進む中，より多くの情報を格納でき，かつ，英数字だけでなく多言語を扱うことができるシンボルの需要が高まっていった．その需要に応える形で開発されたのが二次元シンボルである．

二次元シンボルは図 2.2 に示すように水平・垂直の 2 方向に情報を持たせたメディアである．このことから，水平方向にのみ情報を有するバーコードの数十～数百倍の情報を持たせることが可能であり，この情報密度の高さが二次元シンボルの最大の特徴である．

図 2.2 (a)水平方向にのみ情報を持つバーコード
(b)水平・垂直方向に情報を持つ二次元シンボル

この情報密度の高さから，バーコードでは不可能であった仮名文字や漢字，画像なども表現できる．しかし，シンボルを構成しているバーやセルがバーコードに比べて非常に小さいことから，傷や汚れに弱い．そこで，多くの二次元シンボルは誤り訂正機能を設けることにより，シンボル内の一部が欠けたとしても正常に読み取りを行うことができるように設計されている[4]．ただし，誤り訂正用のデータをシンボルに内蔵することはシンボルサイズの増大を意味し，情報密度や読み取り速度の低下につながる．また，二次元シンボルはバーやセルを増やすことによって情報量を簡易に増加させることができるが，図 2.1(b)のような多量の情報を有するマトリックスは，わずかな歪みや湾曲があるだけで読み取り率が大幅に低下する．そこで，情報量の大きなマトリックスには歪み補正機能が導入される．ただし，これも誤り訂正機能と同様にシンボルサイズの増大による情報密度・読み取り速度の低下につながるという課題がある．

現在，広く利用されている二次元シンボルは PDF417，QR Code，MaxiCode，DataMatrix がある．本節では，これらそれぞれについて，その特徴と利点・欠点について述べる．

2.2.1 PDF417

PDF417(Portable Data File, 4 bar / space, 17 module)は 1989 年にシンボルテクノロジー社によって開発された二次元シンボルである．最大情報量が，英数字：1850 字，数字：2725 桁，バイナリー：1108Byte である．初めてバイナリーの情報化を可能にした二次元シンボルで，音声や画像データも格納可能となっている．図 2.3 に PDF417 の例を示す．

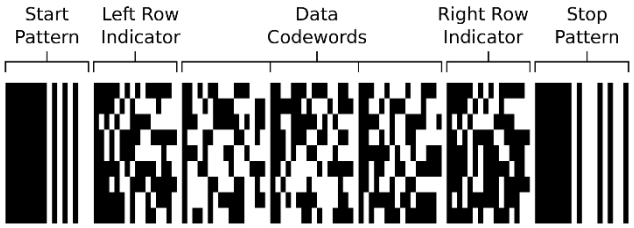

図 2.3 PDF417 の例

PDF417 は，quiet zone(空白領域)，start pattern(開始位置指定領域)，left row indicator(行情報領域)，data codewords(データ領域)，right row indicator(行情報領域)，stop pattern(停止位置指定領域)の 7 種類の領域の組み合わせでできている．図 2.3 の例では，quiet zone・start pattern・left row indicator・right row indicator・stop pattern がそれぞれ 1 領域ずつに加え，data codewords が 3 領域で構成されている．PDF417 は 929 進法のエンコードを用いており，各 data codewords が 0~928 を表している．

読み取り機は上から 1 行ずつ読み込む．隣接する別の行を間違えて読みとらないようにするために，各セルは幅：高＝ 1 ： 3 になるように規格化されている[5]．そのため，幅：高さ＝ 1 ： 1 のセルを使用する二次元シンボルに比べて理論上 3 倍以上の印刷面積になってしまう．実測では PDF417 シンボルは幅：高さ＝ 1 ： 1 のセルを使用する DataMatrix や QR Code と比べて約 4 倍の面積を占めるとの報告もある[6]．また，data codewords の左右に row indicator を設けることにより誤り訂正機能をもたせ，確実な読み取りを可能にしているが，これも印刷面積増大の原因となっている．

ただし, PDF417 はその形状から, 情報量が多いほど, また, 横長になるほど start pattern・left row indicator・right row indicator・stop pattern のオーバーヘッドが減少するため情報化密度は向上する．また，start pattern または stop pattern の一方を取り除くトランケーションシンボルと呼ばれる方法や，セルが幅：高＝ 1 ： 2 の Micro PDF417 が考案されている．

以上のように，PDF417 は確実な読み取りに優れていると言えるが，それと引き換えに情報化密度が比較的低い二次元シンボルと言える．

2.2.2 QR Code

QR Code(Quick Response Code)は，1994 年に株式会社デンソーによって開発された二次元シンボルである．漢字を含む最大 7000 桁の文字が格納可能である．最大の特徴はファインダーパターンとアライメントパターンである．QR Code の例を図 2.4 に示す．

図 2.4 QR Code の例

従来の二次元シンボルは CCD センサーでシンボルを読み取った後，位置・角度・大きさを検出するのに時間がかかるという問題があった．そこで，シンボルの 3 隅にファインダーパターンを配置することで上記問題を解決したのが QR Code である．

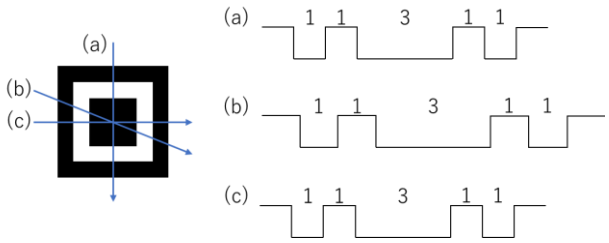

図 2.5 ファインダーパターンの例

図 2.5 に示すように，ファインダーパターンはどの角度から見ても常に 1：1：3：1：1 になる．CCD センサーは 1：1：3：1：1 になる箇所を 3 つ検出し位置関係を特定することで，同時に角度・大きさを検出できる[7]．このファインダーパターンにより，QR Code は他の二次元シンボルより約 20 倍の読み取り速度を誇る．

シンボルの範囲内に一定の間隔で配置されたアライメントパターンは歪み補正機能である．シンボルの外形から推定されるアライメントパターン位置と実際のアライメントパターン位置との差を計算し，マッピングを補正する．

QR Code には 4 種類の誤り訂正レベル(1 シンボルエリアあたり 7%，15%，25%，30%)がある．誤り訂正機能は，各スマッジ/ダメージに応じて実装されており，バーストエラーに強いリード・ソロモン符号が利用されている．

以上のように，QR Code は読み取り補助機能，歪み補正機能，誤り訂正機能などを備え，非常に信頼性の高い二次元シンボルだと言える．ただし，これらの機能を実装することにより，シンボルの印刷面積が増大し，情報化密度の低下につながっていることは否定できない．

2.2.3 MaxiCode

MaxiCode は，1987 年に UPS(United Parcel Service)社が二次元シンボルの高速読み取りを目的として開発した二次元シンボルである．最大の情報量は英数字で 93 字，数字で 138 桁である．シンボル中央に配置されている三重同心円のファインダーパターンと六角形のモジュールが特徴である．汚れや欠けはシンボルの端に集中しやすいという理由から，中央にファインダーパターンを配置している．図 2.6 に MaxiCode の例を示す．

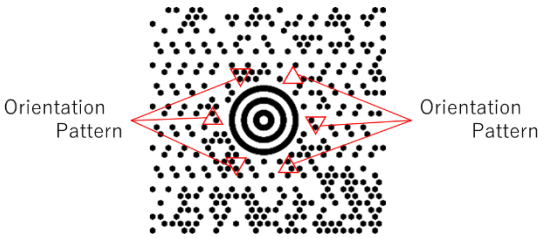

図 2.6 MaxiCode の例

MaxiCode は次の理由により読み取り速度が非常に速い.

- QR Code と同様にファインダーパターンの採用
- 他の二次元シンボルが情報量に合わせてシンボルサイズが増大するのに対して，MaxiCode は情報量に関わらずシンボルサイズを 33 段×30 モジュールを固定化
- モジュールサイズを 1.02±0.12mm(x)×0.88±0.12mm(v)と固定化
- オリエンテーションパターンと呼ばれる傾き補正パターンをファインダーパターンの周りに 6 つ配置
- 最大データ量を低めに制限

以上の特徴より，MaxiCode はデコード時間が非常に短く済むため，読み取り時間が他の二次元シンボルと比較して非常に速い[8].

また, 誤り訂正としてリード・ソロモン符号を採用しており, 標準誤り訂正(Standard Error Collection : SEC)と拡張誤り訂正(Enhanced Error Collection : EEC)がある．SEC は，不読による誤り訂正率：30%，誤読による誤り訂正率 15%となっている．EEC は不読による誤り訂正率：42%，誤読による誤り訂正率：21%となっている．

以上のように，MaxiCode はデコード時間短縮のためにさまざまな工夫がなされており，非常に高速な読み取り速度を実現している．ただし，これらの工夫と引き換えに最大情報量が非常に少ない．そのため，貨物の仕分けや追跡などの，高速読み取りが必要で，かつ，情報量の少ない場面に使用用途が限られる．

2.2.4 DataMatrix

DataMatrix は，1987 年にアイディマトリックス社によって開発された二次元シンボルである．最大情報量は英数字で 2335 字，数字で 3116 字である．その特徴は L 字のアライメントパターンと L 字のタイミングパターンである．その中がデータ領域となっている．

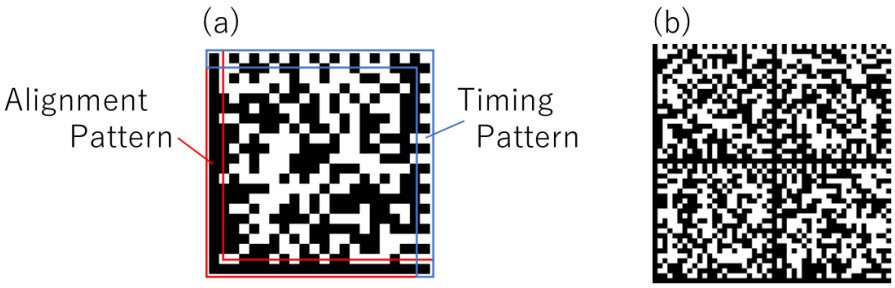

図 2.7 (a)DataMatrix の例 (b)分割された DataMatrix

DataMatrix はアライメントパターンとタイミングパターンの内側が全てデータ領域になっていることから，二次元シンボルでの中で最も小さいシンボルが作成でき，最も情報化密

度が高い．このことから，米国では半導体部品や電子部品のマーキングに利用されている．情報量が多くなるほど僅かな歪みでも読み取り率が低下することから，データセルが 24×24 セル以上になった場合はシンボルを分割し，1 ブロックが 24×24 セル以上にならないようにしている．図 2.7(b)は 4 ブロックに分割されている[9]．

誤り訂正にはリード・ソロモン符号を採用しているが，誤り訂正レベルを任意に設定することはできず，図 2.8 のようにシンボルサイズによって決まる．基本的にシンボルサイズが小さくなるほどレベルは高くなる．

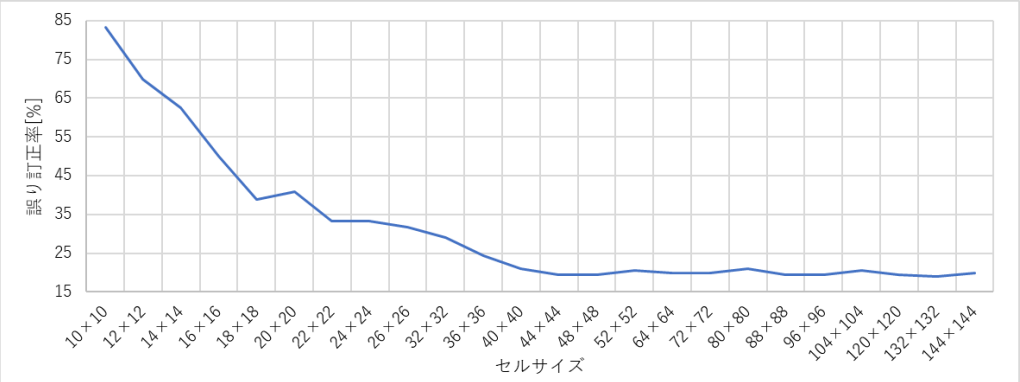

図 2.8 DataMatrix のセルサイズと誤り訂正率の関係[1]

以上のように，DataMatrix は最も情報化密度が高い二次元シンボルであるが，歪みに弱く，歪みの少ない半導体部品や電子部品のマーキングに適している．

2.3　従来手法のまとめ

本章では従来のバーコードシンボル，および，二次元シンボルの概要について述べた．また，現在最もよく利用されている，PDF417，QR Code，MaxiCode，DataMatrix について述べた．

二次元シンボルは，水平・垂直の 2 方向に情報を持たせたメディアであり，水平方向にのみ情報を有するバーコードの数十～数百倍の情報を持たせることが可能である．しかし，シンボルを構成しているバーやセルがバーコードに比べて非常に小さいことから，傷や汚れに弱い．

PDF417 は確実な読み取りに優れていると言えるが，それと引き換えに情報化密度が比較的低い二次元シンボルと言える．

QR Code は読み取り補助機能，歪み補正機能，誤り訂正機能などを備え，非常に信頼性の高い二次元シンボルだと言える．ただし，これらの機能を実装することにより，シンボルの印刷面積が増大し，情報密度の低下につながっていることは否定できない．

MaxiCode はデコード時間短縮のためにさまざまな工夫がなされており，非常に高速な読み取り速度を実現している．ただし，これらの工夫と引き換えに最大情報量が非常に少ない．

そのため，貨物の仕分けや追跡などの，高速読み取りが必要で，かつ，情報量の少ない場面に使用用途が限られる．

DataMatrix は最も情報密度が高い二次元シンボルであるが，歪みに弱く，歪みが少なく印刷面積の少ない半導体部品や電子部品のマーキングに適している．

3　正方形$n \times n$パターン敷き詰めによる二次元コード

3.1　緒言

第 2 章では，従来のバーコードシンボル，および，二次元シンボルについての概要を述べた．二次元シンボルは水平・垂直方向に情報を持つため，情報化密度が高く，小さな面積に多量の情報を格納することができる物理的メディアとして，これからも重要になっていくだろう．また，読み取り速度・情報化密度・信頼性など，二次元シンボルに何を求めるかによってさまざまなシンボルが開発され，棲み分けされている．しかし，多くの二次元シンボルでは読み取りの際ファインダーパターンを読み取ることでシンボルを識別しているため，ファインダーパターンに傷・汚れ・破れがあるとシンボル自体を認識できないという課題がある．また，ファインダーパターンをマトリックス内に配置すると，印刷面積が縮小されるため集積できる情報量が減少してしまうという欠点がある．

本研究では，これらの二次元シンボルに対し，傷や汚れに強く，小さい印刷面積で多くの情報を割り当てることを目的とした，正方形$n \times n$パターン敷き詰め二次元シンボルの提案に向けて基礎的考察を行う．

本章では，本研究で検討する正方形$n \times n$敷き詰めによる二次元シンボルの概要と，二次元シンボルに集積可能な情報量について調査した結果と推測を考察する．それを基に，シンボルを符号語として考えたときの符号化レートについて考察する．また，シンボルを特徴ごとに分類し，シンボルの性質の分析を試みる．

3.2　正方形$n \times n$敷き詰めによる二次元コードの概要

図 3.2 に本手法の2×2パターンの例を示す．白または黒に着色された正方形$n \times n$のパターンを上下左右に敷き詰めることを考え，その一部を$n \times n$パターンで切り取る．このとき，紙面上から切り取られる全ての$n \times n$パターンからは同じ情報を復号する．このように考えたとき，例えば$n = 2$のとき，図 3.3 に示すように2×2のパターンは全部で$2^{2\times2} = 16$通り存在するが，このうち区別できるパターンは 7 通りとなる．ただし，回転・色反転・x 軸対称反転・y 軸対称反転などは考えないものとする．本研究では，同じ読み取り方ができるシンボルを“二次元巡回シフト同値”または“巡回同値”と呼ぶ．例えば図 3.3 では，②のシンボル

は全て巡回同値である．

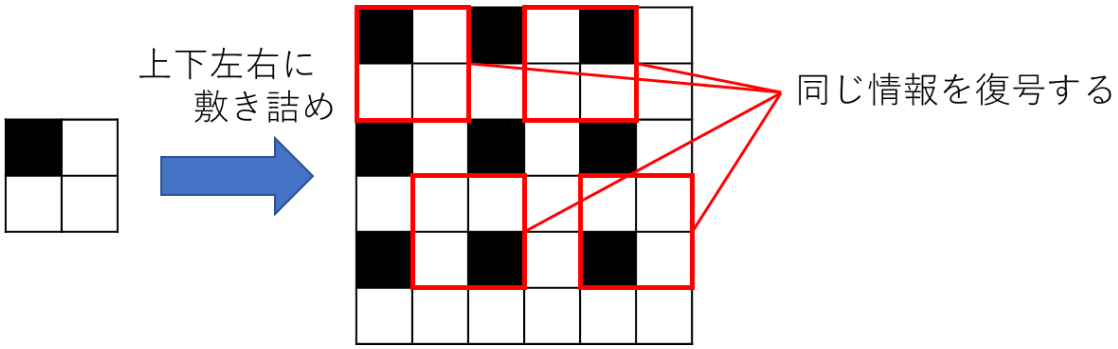

図 3.2 正方形 2×2 敷き詰めの例

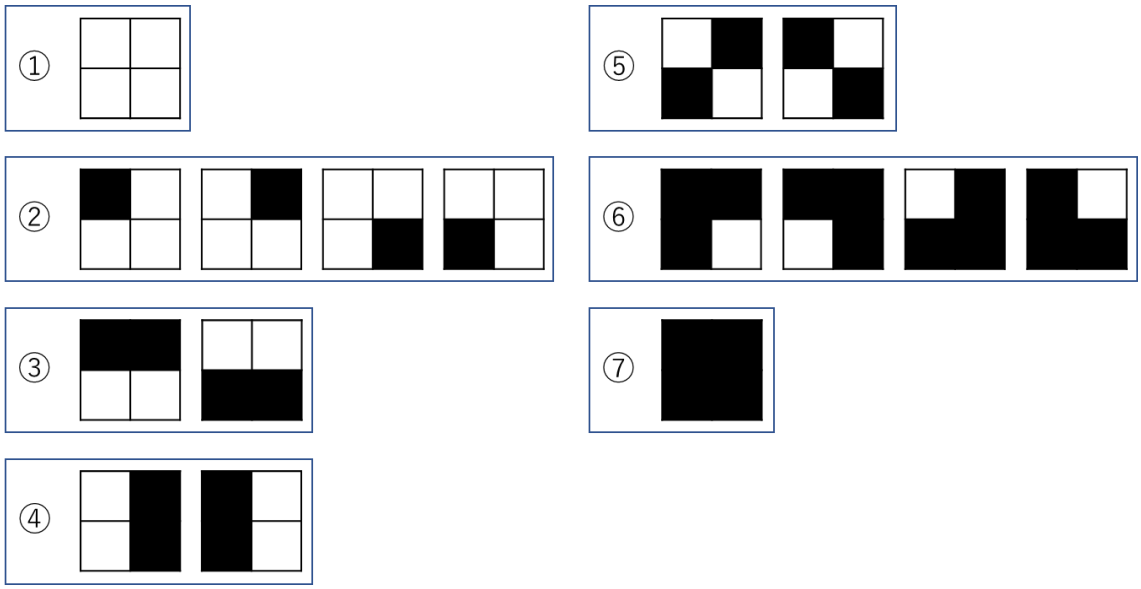

図 3.3 正方形 2×2 敷き詰めの全パターンと区別できるパターン

3.3 パターン数の調査

3.2 節で述べたような条件にて，$n = 1, 2, 3, \cdots$のとき，それぞれ何パターンのシンボルが生成できるかを考える．

3.3.1 巡回同値の親

全$2^{n \times n}$パターンある$n \times n$の正方形を図 3.4 のような行列で考える．a_{xy}には，正方形$n \times n$ 敷き詰めパターンが白ならば 0，黒ならば 1 が入る．

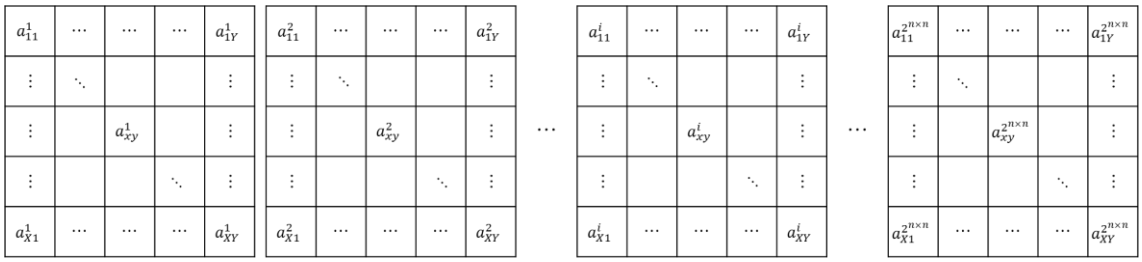

図 3.4　正方形 n × n 敷き詰めパターンを行列に変換して考える

$x = 1,2,\dots,X$, $y = 1,2,\dots,Y$, $X = n$, $Y = n$, $i = 1,2,\dots,2^{n\times n}$である．

ここで，

$$a_{11}^{i}a_{12}^{i}\cdots a_{1Y}^{i}a_{21}^{i}a_{22}^{i}\cdots a_{xy}^{i}\cdots a_{XY-1}^{i}a_{XY}^{i} \tag{3.1}$$

と並べたものを二進数の値として考えたとき，他の巡回同値と比較して一番小さい値のものを本研究では“一番上位の親”と呼ぶ．このとき，巡回同値内で値が小さいほど上位，大きいほど下位とみなす．例えば，図 3.3 に示した2×2のシンボルの例では，②のパターンの中での一番上位の親は左から三番目のシンボルである．

考え方としては次のような 2 段階に分けて親または子の判定およびカウントを行った．
①行列$\left[a_{xy}^{i}\right]$の巡回同値をすべて出し，その中から一番上位の親を探し出す．
②一番上位の親と行列$\left[a_{xy}^{I}\right](=\left[a_{xy}^{1}\right],\left[a_{xy}^{2}\right],\cdots,\left[a_{xy}^{i-1}\right])$を比較し，親か子かを判定する．

それぞれについて詳述する．

①一番上位の親の探索について
A） 図 3.5 のように，$\left[a_{xy}^{i}\right]$の第 X 行を第 1 行に，第 1,2,…X-1 列を第 2,3,…,X 行に，それぞれ移動したものを$\left[a_{xy}^{i}\right]|_{x1,y0}$とする．

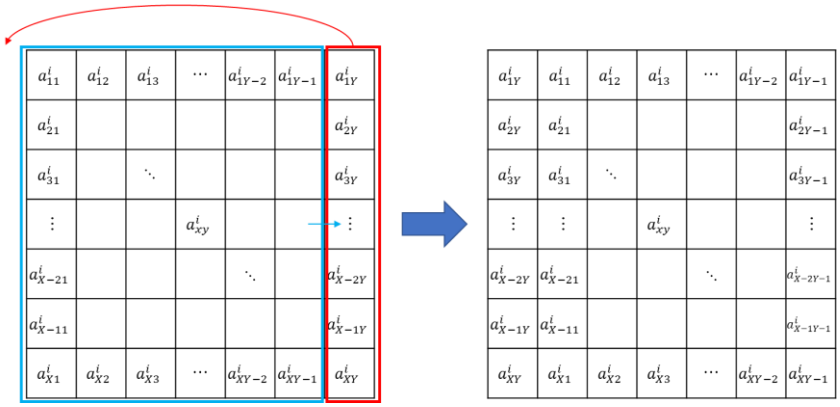

図 3.5 列の移動

B) 図 3.6 のように，$[a_{xy}^i]$の第 Y 列を第 1 列に，第 1,2,⋯,Y-1 列を第 2,3,⋯,Y 列に，それぞれ移動したものを$[a_{xy}^i]|_{x0,y1}$とする．

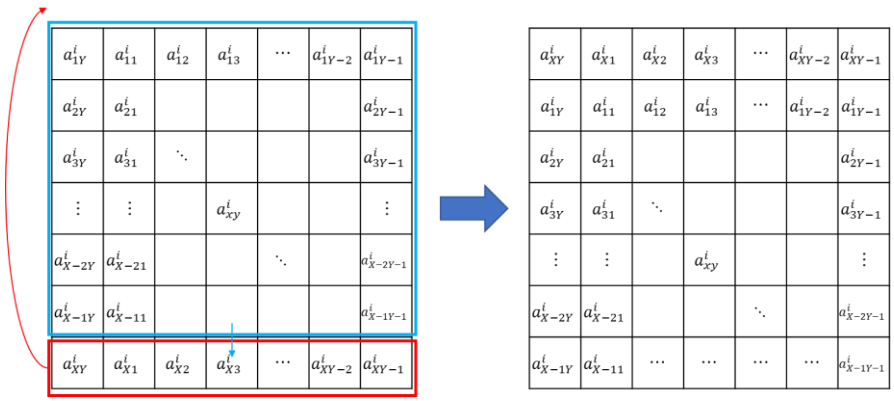

図 3.6 行の移動

C) A), B)の手順に沿って$[a_{xy}^i]|_{x0,y0}$, $[a_{xy}^i]|_{x1,y0}$, $[a_{xy}^i]|_{x2,y0}$, ⋯ , $[a_{xy}^i]|_{xX,yY}$をすべて導出し，それぞれについて式(3.1)のように 2 進数に変換し，最も値の小さいものを一番上位の親$[a_{xy}^i]'$とする。

②親か子かの判定について

①で導出した一番上位の親$[a_{xy}^i]'$を$[a_{xy}^I]$と比較する．$I = 1,2, \dots, i-1$において，一致するものがある場合，$[a_{xy}^i]$をその親の子とする．一致するものがない場合，$[a_{xy}^i]'$を新たな親とする．

以上の操作を$i = 1,2, \dots, 2^{n\times n}$のすべての行列について行い，それぞれを親か子か判定する．

この比較操作を通して得られた親の数が, 3.2 節で述べた区別できるパターンの数となり，

親を含めた子の数が 3.2 節で述べた巡回同値の数となる．正方形$n \times n$敷き詰めによる二次元シンボルにおいて，シンボルを上下左右に敷き詰めてその中から任意に$n \times n$で切り取ったパターンと，シンボルを行列と見なしたときに行および列を 1 行または 1 列ずつ移動させて得られる行列は同じであるといえる．よって，この操作により，正方形$n \times n$敷き詰めによる二次元シンボルの区別できるパターンの数および巡回同値の数を調査できると判断した．

3.3.2　使用言語とプログラミング

使用言語には C 言語を用いた．C 言語には，メモリ領域の管理や，ポインター演算，ビット毎の論理演算など，ハードウェアに密着した処理がしやすいという特徴や，数多あるプログラミング言語の中でも実行速度が速いという特徴がある．プログラムは機械語に翻訳されながら実行されていくインタープリタ方式と，実行される前に機械語に翻訳するコンパイル方式がある．インタープリタ方式は翻訳と実行を同時に行うため，不具合があった場合に確認がしやすいが，実行速度が遅い．一方，コンパイル方式は実行前に翻訳を行うため，不具合があった場合に確認できないが，翻訳と実行の同時処理を行わなくて良いため実行速度が速い．C 言語はコンパイル方式を採用しているため，実行速度が速い．3.3.1 項で述べたように，本研究で提案する二次元シンボルは，総数が$2^{n \times n}$とnの増加とともに爆発的に増加していく．さらに，一つ一つのパターンを$n \times n$回行列の並び替えを行い，他のすべてのパターンと比較処理する．このように，正方形$n \times n$敷き詰めによる二次元シンボルの調査には膨大な計算回数を必要とするため，実行速度の速い C 言語が本研究に適切であると判断した．

プログラムは 3.3.1 項の考え方を基に，図 3.7 に示すフローチャートに則って作成した．

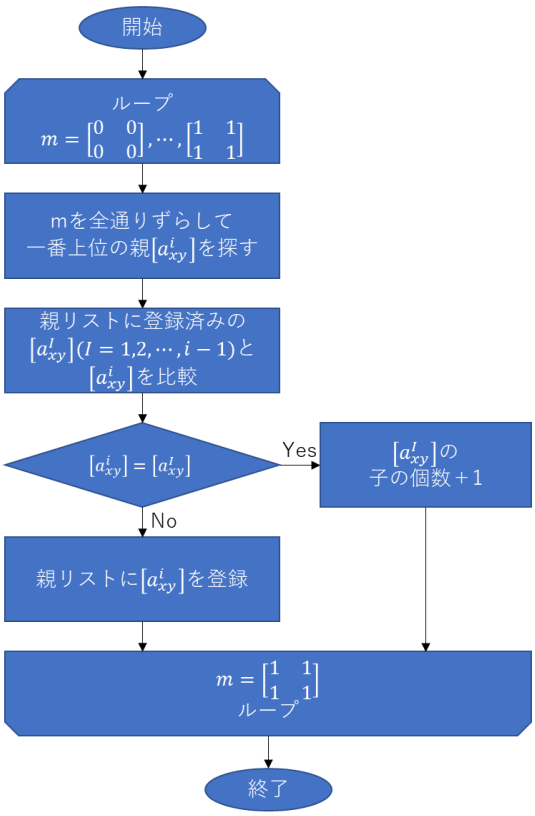

図 3.7　本研究で実行したプログラムのフローチャート

3.4　結果と考察

3.3 節で述べた考え方，および，それを基に組んだプログラムにより，正方形$n \times n$敷き詰めによる二次元シンボルの区別できるパターンの数および巡回同値の数を調査した．ただし，調査することができたのは$n = 1,2,3,4,5$のときのシンボルのみであった．

本研究で使用したコンパイラ Visual Studio は 32 ビットデバッキングである．3.3.2 項で述べたプログラムでは，3.3.1 項で述べた行列$[a_{xy}^{i}]$と行列$[a_{xy}^{I}]$の比較を行う際，行列$[a_{xy}^{i}]$を文字列として$a_{11}^{i}a_{12}^{i}\cdots a_{1Y}^{i}a_{21}^{i}a_{22}^{i}\cdots a_{xy}^{i}\cdots a_{XY-1}^{i}a_{XY}^{i}$のように置き換える工程がある．つまり，$n$行$n$列の行列に対して$n^2$のビット数を消費して計算する．$n = 6$の場合では$6^2 = 36$のビット数を消費する必要があるが，Visual Studio の最大ビット数の 32 ビットを超えており，これを表現することは不可能である．仮に 64 ビットデバッキングのコンパイラを用意することができたとしても，上記理由により$n = 8$までしか調査することはできない．

この問題に加え，本シンボルのパターン数は$2^{n\times n}$とnの増加と共に膨大な数となり，nが大きくなるに従ってプログラムの実行時間が非常に大きくなるという問題がある．

これら 2 つの問題を解決するにはプログラムの実装方法を根本的に見直す必要があったため，$n \geqq 6$の場合についての調査は行わなかった．$n \geqq 6$の親数に関しては，$n = 1,2,3,4,5$で得られた結果を基に推測を行い，その推測から本シンボルの性質や二次元シンボルとし

ての有効性を考察した.

3.4.1　総数と親数の関係

3.3.2 項で作成したプログラムによって$n = 1,2,3,4,5$のときの総数と親数を調査した．また，それらの結果から総数のうちの親数の割合も導いた．表 3.1 にこれらの結果をまとめたものを示す.

nの値の増加に伴い，総数も親数も急激に増加していくことが分かった．ただし，総数のうちの親数の割合(S_{par}/S_{all})はnの増加とともに減少していく傾向にある.

表 3.1　本研究シンボルの総数と親数の関係

$n \times n$	総数 S_{all}	親数 S_{par}	親数/総数 S_{par}/S_{all}
1×1	2	2	1.000000
2×2	16	7	0.437500
3×3	512	64	0.125000
4×4	65536	4156	0.063416
5×5	33554432	1342208	0.040001

表 3.1 より，親数と総数について考察する．総数のうち親数の割合の逆数(以下，“$1/(S_{par}/S_{all})$”)を取った値は$n \times n$の値に近い数になる．これらの関係を表 3.2 に示す．$n > 2$に着目すると，nの増加とともに，$1/(S_{par}/S_{all})$の値は$n \times n$に近づいていく.

表 3.2　(S_{par}/S_{all})の逆数と$n \times n$との一致率

$n \times n$	$1/(S_{par}/S_{all})$	$n \times n$	$1/(S_{par}/S_{all})$と $n \times n$の 一致率
1×1	1.000000	1	1
2×2	2.285714	4	0.571429
3×3	8.000000	9	0.888889
4×4	15.769009	16	0.985563
5×5	24.999427	25	0.999977

$n \geqq 6$においても“n の増加とともに，$1/(S_{par}/S_{all})$の値は$n \times n$に近づいていく”という関係が維持されると仮定すると，

$$\frac{1}{(S_{par}/S_{all})} \cong n \times n$$
$$\Leftrightarrow (S_{par}/S_{all}) \cong \frac{1}{n \times n} \tag{3.2}$$

が成立する．親数は総数S_{all}と総数のうちの親数の割合(S_{par}/S_{all})の積で表されるので，正方形$n \times n$敷き詰めによる二次元シンボルの親数S_{par}は，

$$\begin{aligned} S_{par} &\cong S_{all} \cdot (S_{par}/S_{all}) \\ &\cong \frac{2^{n \times n}}{n \times n} \end{aligned} \tag{3.3}$$

で表すことができる．上式によると親数は，nの増加とともに(S_{par}/S_{all})が減少していくにも関わらず，$n = 6$のときは約5.30×10^7， n =7 のときは約2.34×10^{11}，n=8 のときは約4.50×10^{15}と爆発的に増加していく．

親数，つまり，区別できる数が多いということは，その分の情報をシンボルに割り当てることができるということに繋がる．正方形$n \times n$敷き詰めによる二次元シンボルは，新たな二次元シンボルとしてかなり小さな印刷面積でより多くの識別番号の搭載が可能であることが期待できる．

次に符号化レートについて考察する．符号化レートは符号語数と記号数を用いて，一般に

$$R = \frac{\log_2(\text{符号語数})}{\text{記号数}} \tag{3.4}$$

で表される．正方形$n \times n$敷き詰めによる二次元シンボルにおいては総数S_{all}のうち，親数S_{par}が符号語として使用できる．このとき，S_{par}個の符号語を等確率で用いるとすると，式(3.3)の符号語数Mは親数S_{par}，記号数nはセル数$n \times n$に相当する．したがって，本シンボルにおける符号化レートは式(3.2)および式(3.3)より

$$\begin{aligned} R &= \frac{\log_2 |S_{par}|}{n \times n} \\ &= \frac{\log_2 \left(\frac{2^{n \times n}}{n \times n} \right)}{n \times n} \end{aligned} \tag{3.5}$$

で表される．ただし，送られた符号語を誤りなく推定できるものとする．この式について，$n \to \infty$とすると，

$$
\begin{aligned}
\lim_{n\to\infty} R &= \lim_{n\to\infty} \frac{\log_2\left(\frac{2^{n\times n}}{n\times n}\right)}{n\times n} \\
&= \lim_{n\to\infty} \frac{1}{n\times n}\cdot\{\log_2 2^{n\times n} - \log_2(n\times n)\} \\
&= \lim_{n\to\infty} \frac{1}{n\times n}\cdot\{n\times n - \log_2(n\times n)\} \\
&= \lim_{n\to\infty} \left\{1 - \frac{1}{n\times n}\cdot\log_2(n\times n)\right\} \\
&= 1
\end{aligned}
\tag{3.6}
$$

のようになる．以上より，本シンボルはnが大きくなるにしたがって符号化レートRが 1 に近づいていくことが分かった．これは，1 セルあたり 1 ビットの情報を表すことが可能であることを指し，本シンボルは符号化するにあたっての効率が良いことが示唆される．

従来の二次元シンボルは誤り訂正を考慮するため，符号化レートは 1 より幾らか小さくなる．一方，本シンボルでは誤り訂正を考慮していない．もし，本シンボルに誤り訂正機能を搭載するとなると式(3.5)のような高い符号化レートは得られない可能性がある．しかし，本シンボルはその特性により誤り訂正を考慮する必要性が低い．3.2 節で述べたように，本シンボルはシンボルを紙面の上下左右に敷き詰め，そのうちの$n\times n$セルを任意に読み取る．これはすなわち，印刷面積内に傷・汚れ・破れなどが存在していたとしても，どこか 1 カ所でも$n\times n$セルを読み取ることができればよいということである．本シンボルは誤り訂正を考慮しなくても高確率で読み取りを行うことができる特性から，高い符号化レートが実現できると考えられる．

3.4.2　親の種類に対する子の数

1×1 から 5×5 までの正方形の組み合わせのシンボルについての性質を調べるため，親の種類に対する子の数を調査した．図 3.8，図 3.9 にそれぞれ 2×2 のときおよび 3×3 のときの親の種類に対する子の数の関係を示す．親の種類は，式(3.1)で示したように，行列を文字列に置き換えたもので示している．$n \geqq 4$に関しては親が膨大な数になり，本論文上に図として掲載することが困難だったため，割愛する．また，子の数に対する親の種類数をまとめたものを表 3.3 に示す．

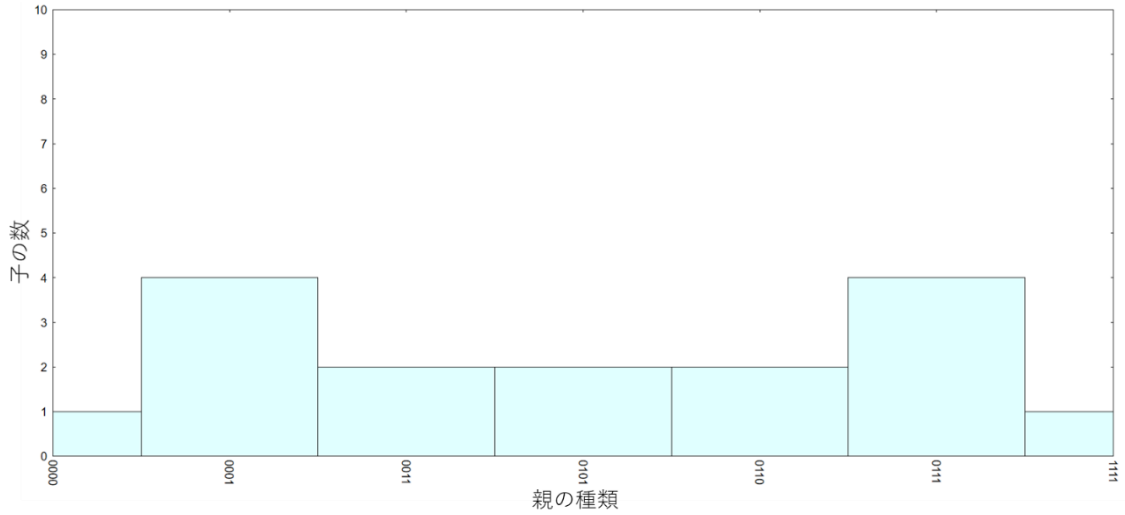

図 3.8　2×2 のときの親の種類に対する子の数

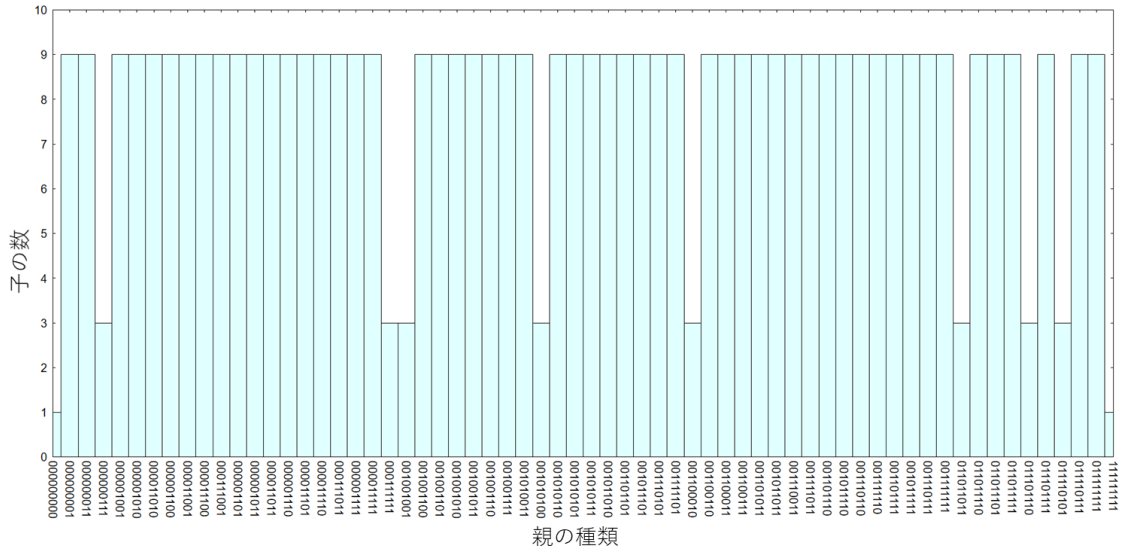

図 3.9　3×3 のときの親の種類に対する子の数

表 3.3 子の数に対する親の数

$n \times n$	子の数N[人]	N人の子を持つ親[人]	割合
1×1	1	2	1.000000
2×2	1	2	0.285714
	2	3	0.428571
	4	2	0.285714
3×3	1	2	0.031250
	3	8	0.125000
	9	54	0.843750
4×4	1	2	4.81×10^{-4}
	2	3	7.22×10^{-4}
	4	20	4.81×10^{-3}
	8	81	1.92×10^{-2}
	16	4050	0.974494
5×5	1	2	1.49×10^{-6}
	5	36	2.68×10^{-5}
	25	1342170	0.999971

全ての要素が 1 の行列および零行列の親だけが一人っ子であり，それ以外の親には必ず兄弟がいることが分かった．また，子の数は必ずn^2の因数になっていることが分かった．$n = 2$のときを除いて，子をn^2人持つ親が圧倒的に多いことが分かった．また，全親数のうち子をn^2人持つ親の数の割合は n の増加とともに増加しており，$n = 5$の時点でn^2人兄弟の子を持つ親がほとんどであった．

これらより，$n = 6$のときは次のように考えられる．子の数 N は 1, 2, 3, 4, 6, 9, 12, 18, 36 となる．また，一人っ子の親は 2 人になる．さらに，子の数が増えるにしたがって親の数も増えていき，36 人兄弟の親がほとんどになる．

3.4.3 子の数に対する親の形

子の数に対して，親がどのような形のシンボルになっているかを調査した．図 3.10，図 3.11 にそれぞれ，2×2 および 3×3 のときの子の数ごとに親の形を分類したものを示す．$n \geqq 4$に関しては親が膨大な数になり，本論文上に図として掲載することが困難だったため，割愛する．多くの親が白黒反転して同じ形になる別の親が存在していることが分かった．例えば，2×2 のときの 4 人兄弟の親の形は 2 種類あるが，一方の親を白黒反転させると，他方の親の形と一致する．この関係は同じ子数の親の間でのみ観察され，異なる子数の親同士

ではこのような関係になるものはなかった．ここで，白黒反転して巡回同値になる関係を“反転同型”と呼ぶことにする．つまり，図 3.10，図 3.11 では，2×2 の 2 人兄弟の親のみが反転同型ではなく，2×2 の一人っ子の親・4 人兄弟の親，3×3 の一人っ子の親・3 人兄弟の親・9 人兄弟の親はすべて反転同型である．反転同型であるか否かは子の数に依るとは限らない．例えば，図 3.12 に示す 4×4 の 4 人兄弟の親や図 3.13 に示す 4×4 の 8 人兄弟の親は反転同型とそうでないものが混在している．

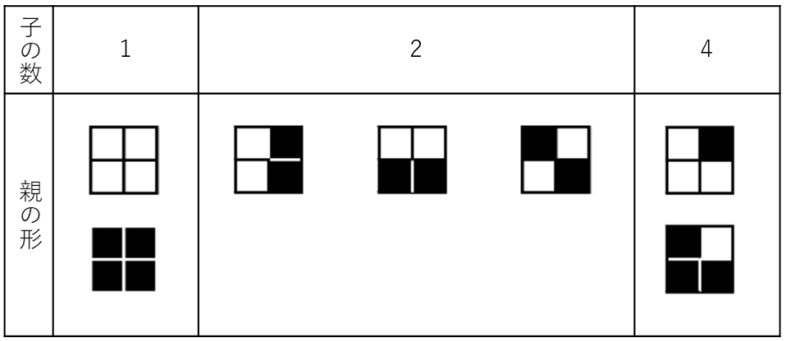

図 3.10　2×2 のときの子の数に対する親の形の分類

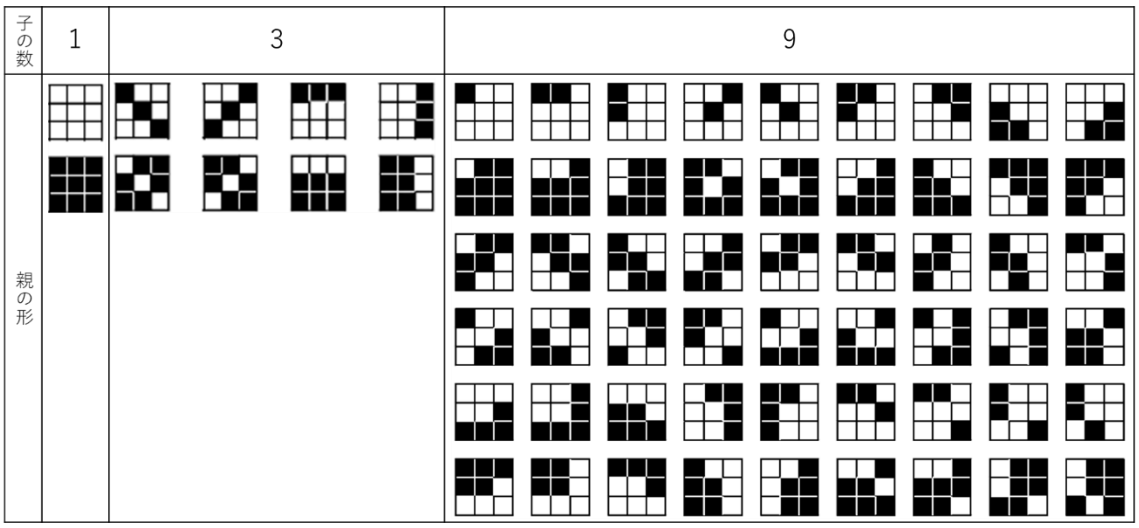

図 3.11　3×3 のときの子の数に対する親の形の分類

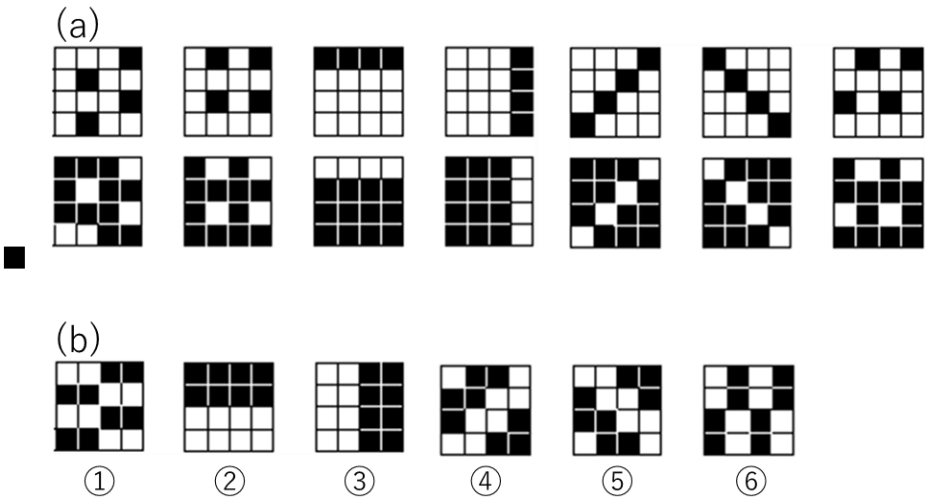

図 3.12　4×4 のときの 4 人兄弟の親の形
そのうち，(a)は反転同型でない親，(b)は反転同型の親

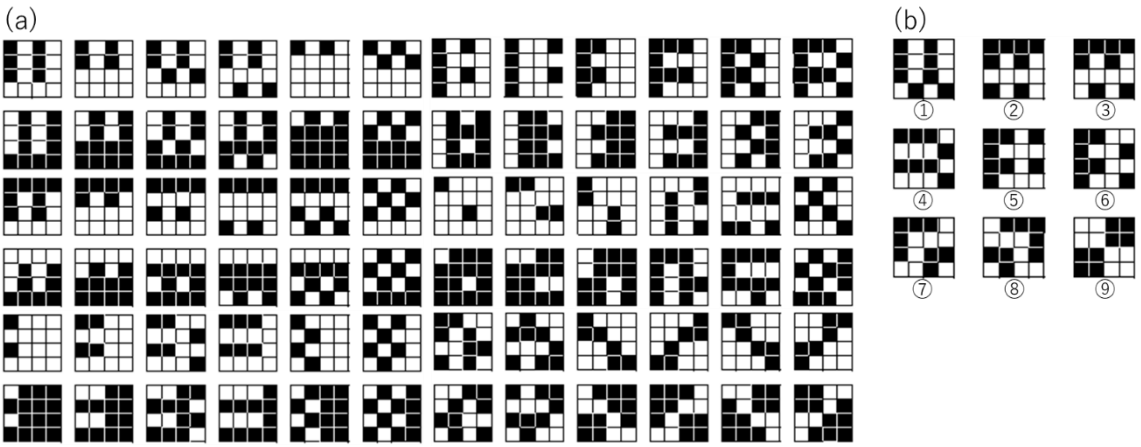

図 3.13　4×4 のときの 8 人兄弟の親の形
そのうち，(a)は反転同型でない親，(b)は反転同型の親

反転同型でない親が生まれる条件について考察する．反転同型でない親が生まれるとき，その親はエッシャー・パターン(Escher Pattern)が成立しており，かつ，基準図形が回転しておらず，さらに，基準図形がn 個のセルによって構築されていることが考えられる．エッシャー・パターンとは，平面充填の一つで，図 3.14 のように一種類の図形で平面が敷き詰められた模様のことである[10]．また，有限種類の図形で隙間なく敷き詰めることを平面充填という．

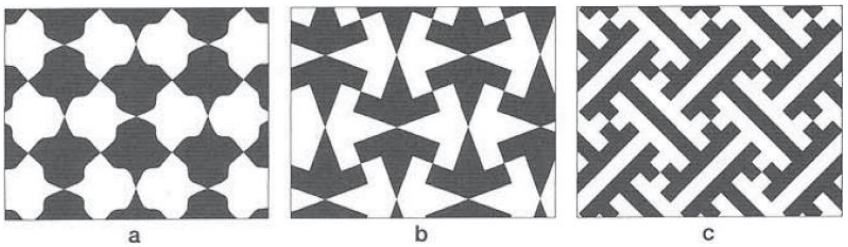

図 3.14 エッシャー・パターンの例[11]

一種類の正多角形による平面充填は正三角形・正方形・正六角形のみが可能であることがピタゴラスによって証明されている．図 3.13(b)⑨がその例であり，正方形 4 つによってシンボルが隙間なく敷き詰められている．図 3.15 に 4×4 の反転同型でない親をエッシャー・パターンに則って基準図形ごとに赤線で区切ったものを示す．分かりやすいように各図形の基準図形の一つを青色で塗りつぶしている．

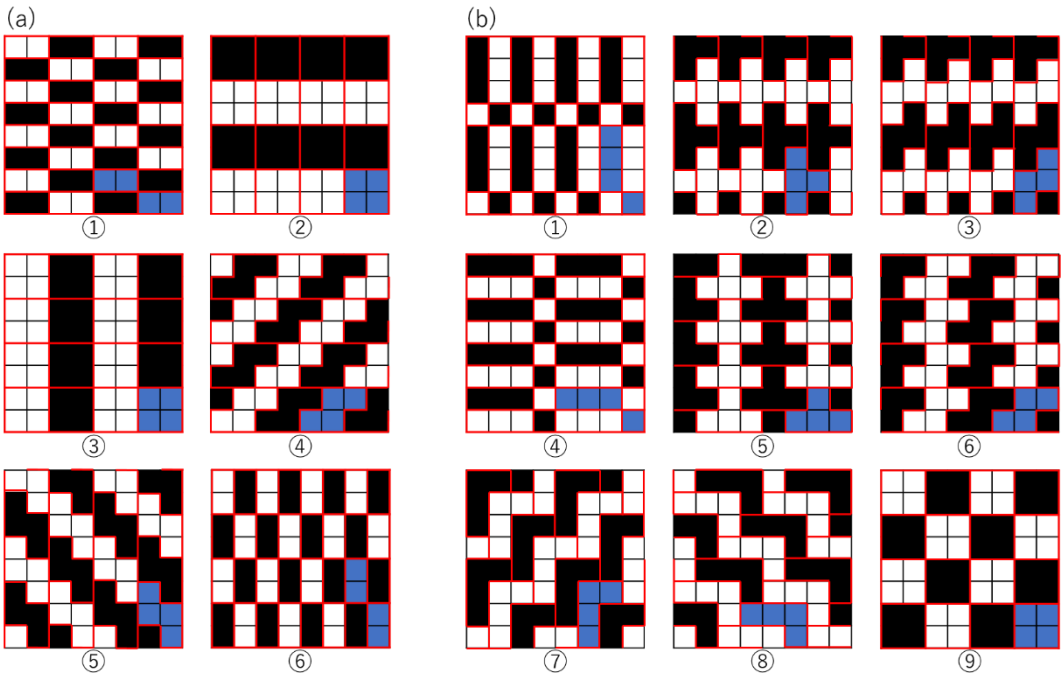

図 3.15 4×4 の反転同型の親を基準図形ごとに赤線で区切ったもの
(a)4 人兄弟を持つ親 (b)8 人兄弟を持つ親

3.2 節で述べた，“正方形$n \times n$敷き詰めによる二次元シンボルでは，シンボルを上下左右に敷き詰めてその中から任意に$n \times n$で切り取ったパターンから元のシンボルと同じ情報を復号できる”という定義を踏まえると，基準図形を回転させずに成立したエッシャー・パターンである反転同型の親は，白色と黒色を反転させても反転前と同じ形となる．

3.5 正方形$n \times n$敷き詰めによる二次元シンボルのまとめ

本章では，新たに正方形$n \times n$敷き詰めによる二次元シンボルの提案に向けて，その性質について調査した．

nの増加とともに総数に対して親になることができるシンボルの割合が減少していくにも関わらず，実際の親数は爆発的に増加していく．親数，つまり区別できる数が多いということは，その分の情報をシンボルに割り当てることができるということに繋がる．正方形$n \times n$敷き詰めによる二次元シンボルは，新たな二次元シンボルとしてかなり小さな印刷面積でより多くの情報の搭載が可能であることが期待できる．

また親数の調査結果から，本シンボルはnが大きくなるにしたがって符号化レートRが 1 に近づいていくと考えられる．これは，1 セルあたり 1 ビットの情報を表すことが可能であることを指し，本シンボルは符号化するにあたっての効率が良いことが示唆される．

4 総括

二次元シンボルは横方向のみに情報を持つバーコードなどの一次元シンボルに対し，水平方向と垂直方向に情報を持つシンボルであるため，小さい面積に印刷可能，かつ，集積できる情報量が多いことが特徴である．近年では二次元シンボルのマトリックス内に位置検出のためのファインダーパターン（切り出しシンボル）と呼ばれる特徴的な図形が配置され，シンボルの位置や大きさ，傾きを検出でき，360°全方向から読み取ることができるものも実用化されている．一方，ファインダーパターンをマトリックス内に配置すると，印刷面積が縮小されるため集積できる情報量が減少してしまうという欠点がある．

本研究では，ファインダーパターンをマトリックス内に配置せず，正方形$n \times n$マトリックスを上下左右に大量に敷き詰めることによって位置検出を不要とする二次元シンボルの表示方式の提案に向けて，その性能を調査した．本手法は，ファインダーパターンがないために簡易なマトリックスにすることができる．

本シンボルは，nの増加とともに総数に対して親になることができるシンボルの割合が減少していくにも関わらず，実際の親数は爆発的に増加していく．親数が多いということは，その分の情報をシンボルに割り当てることができるということに繋がる．このことから，$n \times n$正方パターン敷き詰めによる二次元シンボルは，新たな二次元シンボルとして多くの情報の搭載が可能であることが期待できる．

また，本研究での親数の推測を基に，シンボルの符号化レートについて考察した結果，nの増加と共に符号化レートは 1 に近づくことが分かり，本シンボルは符号語として使用するにあたって効率が良いと予想できる．

正方形$n \times n$パターンの敷き詰めによる二次元コードは，ファインダーパターンを排した構造とすることで，従来の二次元シンボルの欠点であったファインダーパターンの汚れ・欠

損により読み取り不可となる現象を克服できる可能性を秘めている．本シンボルを実用化段階まで深く考察することで，新たな二次元シンボルとして，従来の二次元シンボルより多くの場面で活用できると考えられる．

謝辞

本研究を行うにあたって，丁寧なご指導を賜りました指導教員の西新幹彦准教授に深く感謝申し上げる．

また，多くのご意見を頂いた西新研究室の皆様に深く感謝申し上げる．

参考文献

[1] 平本純也，「バーコードの知識」，日本工業出版, 1991.

[2] Y. Cai, P. Li, X.W. Li, J. Zhao, H. Chen, Q. Yang, and H. Hu. “Converting Panax ginseng DNA and chemical fingerprints into two-dimensional barcode”, Journal of Ginseng Research, no.41, pp.339-346, Jul. 2016.

[3] Y. Cai, X.W. Li, M. Li, X.J. Chen, H. Hu, J.Y. Ni, and Y.T. Wang, “Traceability and Quality Control in Traditional Chinese Medicine: From Chemical Fingerprint to Two-Dimensional Barcode”, Evidence-Based Complementary Alternative Medicine, ID 251304, pp.1-6, Oct. 2014.

[4] 標準化研究学会，「QRコードのおはなし」，日本規格協会, 2002.

[5] ISO / IEC 15438:2015. Information technology — Automatic identification and data capture techniques — PDF417 bar code symbology specification.

[6] *Using Barcodes in Documents – Best Practices*, Tampa, FL: Accusoft, (2007).

[7] T. J. Soo, “QR Code”, Synthesis Journal, 2008.

[8] ISO / IEC 16023:2000. Information technology — International symbology specification — MaxiCode.

[9] ISO / IEC 16022:2006 Information technology — Automatic identification and data capture techniques — Data Matrix bar code symbology specification.

[10] E.H. Gombrich. “The Sense of Order : a Study in the Psychology of Decorative Art”, Oxford: Phaidon, 1979.

[11] F. Shin. “Considerations of Penrose’s nonperiodic Patterns and Escher’s Petterns”, Bulletin of Japanese Society for the Science Desgin, vol.47 no.5 pp.29-38, Nov. 2000.

付録A ソースコード

A.1 親か子か判定するプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#define SIZE 4
#define SIZE2 (SIZE * SIZE)
#define POWSIZE2 (1 << SIZE2)

main()
{
        int n = SIZE;
        int N = n * n;
        int nn2 = 1 << N;
        int p[SIZE][SIZE];
        int i, j;
        int blist[SIZE][SIZE];

        for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++) {
                        blist[i][j] = 0;
                }
        }
        char* count;
        count = (char*)calloc(POWSIZE2, sizeof(char));

        /* 二次元配列parlistをダブルポインタで取得*/
        char** parlist = NULL;
        parlist = (char**)calloc(POWSIZE2, sizeof(char*));
        if (parlist == NULL) {
                return 1;
```

```
}
int k;
int pi = 0;
/* 1ずつカウントアップ*/
for (;;) {
	printf("%s", ".");
	for (i = 0; i < n; i++) {
		for (j = 0; j < n; j++) {
			printf("%d", blist[i][j]);
		}
	}
	putchar('¥n');

	/* ここで比較用配列Pを作成(１埋めしておく) */
	for (i = 0; i < n; i++) {
		for (j = 0; j < n; j++) {
			p[i][j] = 1;
		}
	}

	int c, r, i_, j_;
	int m[SIZE][SIZE];
	/* 全通りずらす*/
	for (c = 0; c < n; c++) {
		for (r = 0; r < n; r++) {
			for (i = 0; i < n; i++) {
				for (j = 0; j < n; j++) {
					i_ = (i - r + n) % n; /*縦にずらす*/
					j_ = (j - c + n) % n; /*横にずらす*/
					m[i][j] = blist[i_][j_];
				}
			}
			/* 作ったm配列とＰ配列を比較して小さいほうをPに保管*/
			int k, l;
			int flag = 0;
			for (k = 0; k < n; k++) {
```

```
                        for (l = 0; l < n; l++) {
                                if (m[k][l] < p[k][l]) {
                                        for (i = 0; i < n; i++) {
                                        for (j = 0; j < n; j++) {
                                                p[i][j] = m[i][j];}
                                        }
                                        flag = 1;
                                        }
                                else if (m[k][l] > p[k][l]) {
                                        flag = 1;
                                }
                                if (flag) {
                                        break;
                                }
                        }
                        if (flag) {
                                break;
                        }
                }
        }
}

/* 一番若い親pを二次元親リストparlistに入れる*/
int ploop;
int pfrag;
pfrag = 0;
for (k = 0; k < pi; k++) {
        /* parlistに既に入っているか調べる*/
        ploop = 0;
        for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++) {
                        if (p[i][j] == parlist[k][ploop]) {
                                pfrag = 1;
                        }
                        else {
                                pfrag = 0;
```

```
                                        break;
                        }
                        ploop++;
                }
                if (pfrag == 0) {
                        break;}
                }
                if (pfrag) {
                        break;
                }
        }
        /* 登録済だったらカウントする*/
        if (k < pi) {
                count[k]++;
        }
        /* ない場合は登録する*/
        else {
                parlist[pi] = (char*)calloc(SIZE2, sizeof(char));
/*メモリ不足で指定サイズ分のメモリが確保できないときNULL*/
                if (parlist[pi] == NULL) {
                        return -1;
                }
                ploop = 0;
                for (i = 0; i < n; i++) {
                        for (j = 0; j < n; j++) {
                                parlist[pi][ploop] = p[i][j];
                                ploop++;
                        }
                }
                pi++;
                count[k]++;
        }
/* カウントアップ*/
        for (i = 0; i < N; i++) {
                if (blist[i / n][i % n] == 0) {
                        blist[i / n][i % n] = 1;
```

```
                                        break;
                                }
                                blist[i / n][i % n] = 0;
                        }
                        if (i == N) {
                                break;
                        }
                }
}
int num[SIZE2] = { 0 };
int count_num[SIZE2] = { 0 };
int jcount = 0;
for (i = 0; i < pi; i++) {
        for (j = 0; j < jcount; j++) {
                if (num[j] == count[i]) {
                        count_num[j]++;
                        break;
                }
        }
        if (j == jcount) {
                num[jcount] = count[i];
                count_num[jcount]++;
                jcount++;
        }
}
for (i = 0; i < jcount; i++) {
        printf("%d %d¥n", num[i], count_num[i]);
}
for (i = 0; i < pi; i++) {
        for (j = 0; j < N; j++) {
                printf("%d", parlist[i][j]);
        }
        printf(" %d¥n", count[i]);
}
putchar('¥n');
printf("%d¥n¥n", pi);
```

```
	/* ファイルに書き込む*/
	FILE* fp;
	errno_t error; //fopen_sでは戻り値errno_t
	error = fopen_s(&fp, "result5.txt", "w");
	for (i = 0; i < pi; i++) {
		for (j = 0; j < N; j++) {
			fprintf(fp, "%d", parlist[i][j]);
		}
		fprintf(fp, " %d¥n", count[i]);
	}
	fclose(fp);
	FILE* cfp;
	errno_t cerror;
	cerror = fopen_s(&cfp, "result5_5.txt", "w");
	int i_count = 0;
	for (i = 0; i < pi; i++) {
		if (count[i] == 5) {
			for (j = 0; j < N; j++) {
				fprintf(cfp, "%d", parlist[i][j]);
				if (j % SIZE == (SIZE - 1)) {
					fprintf(cfp, "¥n");
				}
			}
		fprintf(cfp, "¥n%d¥n", i);
		fprintf(cfp, "%d¥n¥n", i_count);
		i_count++;
		}
	}
	fclose(cfp);
	free(count);
	free(parlist);
	return 0;
}
```