CF-004

3 段階進化戦略による魔方陣生成
Three-stage Evolutionary Strategy for Magic Squares

竹味 和輝[1)]　佐久間 拓人[2)]　加藤 昇平[2)]
Kazuki Takemi　Takuto Sakuma　Shohei Kato

1　はじめに

魔方陣は紀元前より多くの人に娯楽数学の一つとして取り組まれてきたテーマである．魔方陣の総数は 3 次魔方陣が 1 種類，4 次魔方陣が 880 種類，5 次魔方陣が 275,305,224 種類 [1] と知られており，6 次以上の魔方陣では正確な数が算出できないほど膨大な種類数が存在する．任意の大きさの魔方陣を作ることは複雑な組合せ最適化問題となるため既存手法にはルールベースに則った手法が多い [2]．それゆえ，特定の条件を満たす特殊例の魔方陣がほとんどであり，多種多様な魔方陣を生成することが困難である．

そこで，ランダムに数を配置して魔方陣を生成する方法が考えられるが，1 から n^2 までの数字の順列をランダムに配置した時に魔方陣の条件を満たす確率は著しく低い．生成確率は 3 次魔方陣では 2.204×10^{-5} であるが次数が 1 つ大きくなるにつれて 10^{-6} から 10^{-9} ほど下がり 10 次，30 次魔方陣の生成確率はそれぞれ 2.069×10^{-47}，7.78×10^{-213}[3] よりランダムに配置して偶然魔方陣となることはほぼないと言えるため，次に進化計算を用いた生成について考える．

進化的アルゴリズムを用いた魔方陣の生成としては Xie & Kang[4] が 2 段階による生成手法を提案している．魔方陣から対角線の制約条件を外した半魔方陣を生成するまでを第 1 段階，半魔方陣から魔方陣を生成する過程を第 2 段階としているが，2 段階に分けず 1 段階で生成を試みた場合は 10×10 の魔方陣ですら生成が困難となる．これは定和として揃っている行や列を崩しながら次世代を生成してしまうため，魔方陣の条件を満たすまで膨大な世代数が必要になるためである．そのため，適切に段階を分けて生成することが魔方陣を構築する上では重要となる．

また，先行研究 [4] では半魔方陣を生成する過程で多くの時間を要しており，その中の大半を占めていたのが半魔方陣の生成終盤において適用するヒューリスティック手法の局所修正 (Local Rectification) である．局所修正では各行・各列に対して交換条件を満たしているか探索するため膨大な時間がかかる．そこで．本研究ではこの探索回数を大幅に減らすために先行研究 [4] で提案されていた 2 段階の前段階として，「各行の和を等しくする」という 1 段階を加えた進化戦略 [5] に基づく 3 段階の魔方陣生成を提案する．さらに，半魔方陣から魔方陣を生成する段階は，先行研究 [4] のアルゴリズムを再現して実装したところ多くの解が局所解に陥り魔方陣を生成することが困難であった．そのため，局所修正の適用と進化戦略の変更のための閾値を変更し改善した．本研究では実験において，1 辺の大きさが偶数の魔方陣及び半魔方陣の生成に要する時間の短縮を確認した．

1)　名古屋工業大学 工学部 情報工学科
Dept. of Computer Science, Nagoya Institute of Technology
2)　名古屋工業大学 大学院工学研究科 工学専攻
Dept. of Engineering, Graduate School of Engineering, Nagoya Institute of Technology

2　魔方陣

Definition 2.1. 魔方陣

次の条件を満たすものを魔方陣という．

- 各行・各列・対角線 の数字の和が定和に等しい
- 1 から n^2 までの数字を過不足なく用いる

定和とは，魔方陣において一定となる 1 列の和の値である．

n 次 $(n\times n)$ の魔方陣を行列 M，定和を c と表すと，魔方陣は次のように定式化できる．

$M=(a_{ij})_{n\times n}$, where $a_{ij}\in\{1,2,3,\cdots,n^2\}$, $a_{ij}\neq a_{kl}$, $i\neq k$ or $j\neq l$

$\sum_{i=1}^{n}a_{ij}=c$, $\sum_{j=1}^{n}a_{ij}=c$, $\sum_{i=1}^{n}a_{ii}=c$, $\sum_{i=1}^{n}a_{i,(n-i+1)}=c$, where $c=\frac{n(n^2+1)}{2}$, $i,j=1,2,3,\cdots,n^2$

また，列の数字の和が定和と一致しない行数，列数をそれぞれ n_{row},n_{col} と表す．

Definition 2.2. 半魔方陣

魔方陣の制約条件のうち，「対角線が定和に等しい」$(\sum_{i=1}^{n}a_{ii}=c,\ \sum_{i=1}^{n}a_{i,(n-i+1)}=c)$ という条件を省いた下記の条件を満たす方陣を半魔方陣と言う．

- 各行・各列 の数字の和が定和に等しい
- 1 から n^2 までの数字を過不足なく用いる

3　提案手法

進化戦略

親集合を U，子集合を Θ，母集合を $\Pi=U\cup\Theta$，$|U|=\mu$，$|\Theta|=\lambda$ ($|U|$，$|\Theta|$ はそれぞれ U,Θ の個体数) とすると，今回用いる 2 つの進化戦略 (ES) 手法は以下のように表せる．

- $(\mu+\lambda)$-ES
  母集合 $\Pi=U\cup\Theta$ の中から最も適応度の高い個体を次世代の親とする．
- (μ,λ)-ES
  子集合 Θ の中から最も適応度の高い個体を次世代の親とする．

3.1　3 段階構築

第 1 段階で各行の和が定和になるように作成し，第 2 段階で半魔方陣，第 3 段階で魔方陣を完成させる 3 段階で構築する．

第 1 段階を追加したために，余分に時間がかかってしまっては 3 段階に分ける意味がない．そのため，第 1 段階の作成は高速にできるように，列方向に隣接する 2 つの和がどの行も等しくなるようにペアを組んでいくように以下のように定めた．

i 行 k 列の数字について

$$\begin{cases} a_{ik} = (k-1)n + i & (k \text{ が奇数}) \\ a_{ik} = kn + 1 - i & (k \text{ が偶数}) \end{cases} \tag{1}$$

j 列と $j+1$ 列 ($j \equiv 1 (\mathrm{mod}\ 2)$) について，全ての行において $a_{ij} + a_{i,(j+1)}$ の和は等しくなる．

$a_{ij} + a_{i,(j+1)} = ((j-1)n + i) + (j+1)n + 1 - i = 2jn + 1$

また，この方法は魔方陣の次数 n が偶数のときのみ有効であるため，本稿では n が偶数の場合のみを扱う．

3.2 適応度関数

適応度を f とすると，適応度関数は先行研究 [4] と同じく，各行・各列・対角線において定和との差の絶対値を足し合わせたものとし，次のように定義できる．

Definition 3.1. 適応度関数

$$f(M) = \begin{cases} \sum_{i=1}^{n} \left| c - \sum_{k=1}^{n} a_{ik} \right| + \sum_{j=1}^{n} \left| c - \sum_{k=1}^{n} a_{kj} \right| \\ \qquad (if(n_{row} + n_{col}) > 0). \\ \sum_{i=1}^{n} \left| c - \sum_{k=1}^{n} a_{ik} \right| + \sum_{j=1}^{n} \left| c - \sum_{k=1}^{n} a_{kj} \right| - \\ \qquad \left| c - \sum_{i=1}^{n} a_{ii} \right| - \left| c - \sum_{i=1}^{n} a_{i,(n-i+1)} \right| \\ \qquad (if(n_{row} + n_{col}) = 0). \end{cases} \tag{2}$$

3.3 突然変異

行内交換：第 2 段階

$n_{row} + n_{col} > 0$ のとき

1. ランダムに 1 行を選択
2. 選択した行内の 2 マスを次の 2 手法のいずれかにしたがって交換 (手法の選択は等確率)

・定和になっていない列のマス同士を交換
・2 マスのうちいずれか一方が定和でない列の交換

行・列ごと交換：第 3 段階

$n_{row} + n_{col} = 0$ のとき

ランダムに 2 行もしくは 2 列を選択して，行・列ごとを交換.

3.4 局所修正

魔方陣を生成していく過程では終盤になればなるほど解の改善は難しくなる．そこで，終盤にヒューリスティックな手法として以下で定義する局所修正を適用する．ここでは，すべての行・列の探索を行い一部のマスの交換により定和になる列が存在した場合は．該当するマスを交換することを局所修正という．

列方向局所修正

1 マス同士の交換として，a_{sk} と $a_{sl} (k \neq l)$ を交換することにより k,l 列がともに定和となる条件を以下に示す.

$$\sum_{i=1}^{n} a_{ik} - c = c - \sum_{j=1}^{n} a_{jl} = a_{sk} - a_{sl} \tag{3}$$

1 マス同士の交換と同様に，a_{sk}, a_{tk} と $a_{sl}, a_{tl} (k \neq l, s \neq t)$ の 2 マスずつ交換することにより k,l 列がともに定和となる条件を次に示す．(行方向も同様)

$$\sum_{i=1}^{n} a_{ik} - c = c - \sum_{j=1}^{n} a_{jl} = (a_{sk} + a_{tk}) - (a_{sl} + a_{tl}) \tag{4}$$

先行研究 [4] では, 式 3,4 を用いた局所修正を提案していた．しかし，この手法では交換後に k,l 列の和がともに定和になる場合にしか交換が行われない．そのため，交換後にいずれかの列が定和になる場合でも交換することで解の改善が進むと考え，本研究では以下の条件を満たす場合に交換する．

次の条件を満たすとき，a_{sk} と a_{sl} を交換

$$\sum_{i=1}^{n} a_{ik} - c = a_{sk} - a_{sl} \wedge \sum_{j=1}^{n} a_{jl} - c = a_{sk} - a_{sl} \tag{5}$$

次の条件を満たすとき，a_{sk}, a_{tk} と a_{sl}, a_{tl} を交換

$$\begin{aligned} &\sum_{i=1}^{n} a_{ik} - c = (a_{sk} + a_{tk}) - (a_{sl} + a_{tl}) \\ \wedge &\sum_{j=1}^{n} a_{jl} - c = (a_{sk} + a_{tk}) - (a_{sl} + a_{tl}) \end{aligned} \tag{6}$$

また，3 マス同士以上の交換は探索に膨大な時間がかかるため先行研究 [4] と同様，局所修正に加えない.

対角線局所修正

先行研究 [4] では 5 つの局所修正 (表 1) を定義し，条件を満たすときに該当するマスや行・列を交換していた．本研究でも，同様の局所修正を採用する.

3.5 アルゴリズム

魔方陣を構築するアルゴリズムでは，交叉や組換え演算子を用いないため，親の数はアルゴリズムの効率に影響を与えない．そこで，本研究では親個体を 1 つとし各世代で 10 個の子個体を生成する．図 1 にアルゴリズムの全容を示す.

Step1 : 初期化

全ての行において 1 行の和が定和となるように 1 から n^2 までの数字を過不足なく埋める．(第 1 段階)

Step2 : メインアルゴリズム

$n_{row} + r_{col} > 0$ つまり第 2 段階のとき， ランダムで 1 行を選択した上でその行内の 2 マスを交換する．その後，最良個体の適応度 が $0.25n^3$ 以下のとき，行・列方向の局所修正を適用する.

$n_{row} + r_{col} = 0$ つまり第 3 段階のとき， ランダムで 2 行もしくは 2 列を選択し 行・列ごとに交換する．その後，最良個体の適応度 が $0.45n^2$ 以下のとき，対角線局所修正を適用する.

Step3 : 子の生成

$n_{row} + r_{col} > 0$ つまり第 2 段階のとき，最良個体の適応度が $0.25n^3$ を超えるとき (μ, λ)-ES を適用し，それ以外は $(\mu + \lambda)$-ES を適用する.

$n_{row} + r_{col} = 0$ つまり第 3 段階のとき，最良個体の適応度が $0.45n^2$ を超えるとき (μ, λ)-ES を適用し，それ以外は $(\mu + \lambda)$-ES を適用する.

Step4 : 終了条件

最良個体の適応度が 0 になったら終了し，そうでなければ Step2 へ進む.

表 1: 対角線局所修正 ($1 \le i \le n,\ 1 \le j \le n,\ i \ne j$)

	交換対象	交換条件
1	a_{ii} と a_{ji}，a_{ij} と a_{jj} を交換	$(a_{ii}+a_{ij}=a_{ji}+a_{jj}) \wedge ((a_{ii}+a_{jj})-(a_{ij}+a_{ji})=\sum_{k=1}^{n} a_{kk}-c)$
2	a_{ij} と $a_{n-j+1,j}$，$a_{i,n-i+1}$ と $a_{n-j+1,n-i+1}$ を交換	$(a_{ij}+a_{i,n-i+1}=a_{n-j+1,j}+a_{n-j+1,n-i+1}) \wedge ((a_{i,n-i+1}+a_{n-j+1,j})-(a_{ij}+a_{n-j+1,n-i+1})=\sum_{k=1}^{n} a_{n-k+1,k}-c)$
3	i 行と j 行を交換	$((a_{ii}+a_{jj})-(a_{ij}+a_{ji})=\sum_{k=1}^{n} a_{kk}-c) \wedge ((a_{i,n-i+1}+a_{j,n-j+1})-(a_{i,n-j+1}+a_{j,n-i+1})=\sum_{k=1}^{n} a_{n-k+1,k}-c)$
4	i 列と j 列を交換	$((a_{ii}+a_{jj})-(a_{ij}+a_{ji})=\sum_{k=1}^{n} a_{kk}-c) \wedge ((a_{n-i+1,i}+a_{n-j+1,j})-(a_{n-j+1,i}+a_{n-i+1,j})=\sum_{k=1}^{n} a_{n-k+1,k}-c)$
5	i 行と $(n-i+1)$ 行を交換	$(a_{ii}+a_{n-i+1,n-i+1})-(a_{i,n-i+1}+a_{n-i+1,i})=\sum_{k=1}^{n} a_{kk}-c=c-\sum_{k=1}^{n} a_{n-k+1,k}$

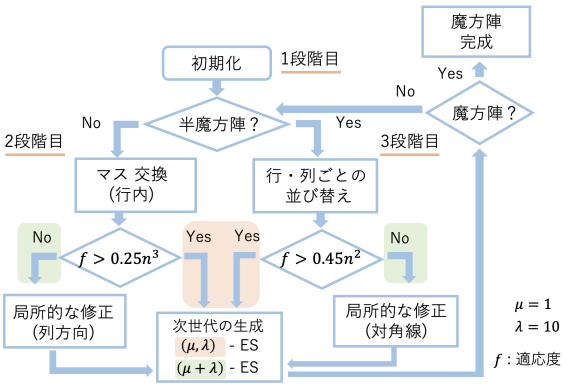

図 1: 3 段階進化戦略

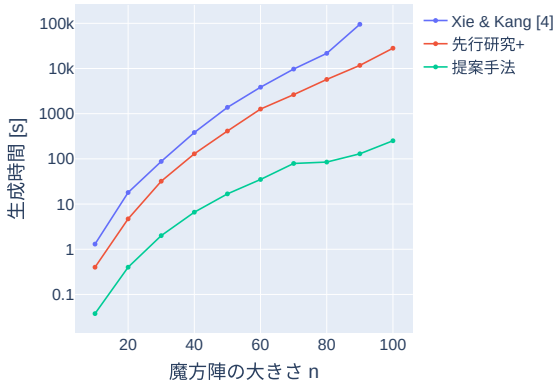

図 2: 第 1・2 段階の平均生成時間

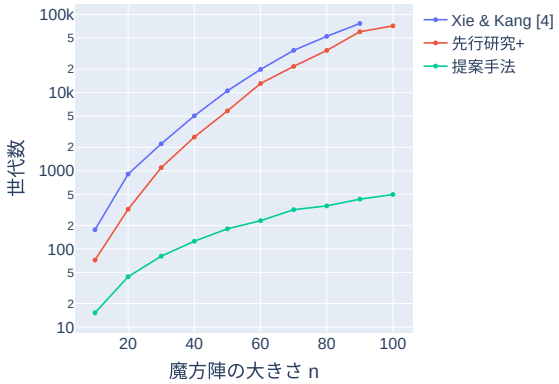

図 3: 第 1・2 段階の平均世代数

4 実験

4.1 実験方法

10×10 から 100×100 までの魔方陣を 10 個ずつ生成し，その生成時間と平均世代数について比較する．実行環境は AMD Ryzen Threadripper 3970X 32-core processor を用いた．

第 1・2 段階

Xie & Kang の先行研究 [4] の再現手法，先行研究の局所修正のみを改善した手法 (以降先行研究+とする)，提案手法の 3 手法で半魔方陣を生成したときの生成時間，平均世代数を比較する．第 1 段階の生成時間は第 2 段階の生成時間と比べると無視できるほど小さいため，1・2 段階はまとめて実験する．

第 3 段階

Xie & Kang の先行研究 [4] では，局所修正の適用と進化戦略の変更のための閾値を 100 としていたが，この閾値では魔方陣が生成できない試行が多く存在した．そのため，提案手法ではこの閾値のみ $0.45n^2$ へ変更し，先行研究の再現手法との生成時間，平均世代数の違いを比較する．

4.2 実験結果

第 1・2 段階

表 2 に各手法における実験結果を示す．提案手法による生成時間を Xie & Kang の先行研究 [4] と比較すると，$n = 60$ のとき約 110 分の 1，$n = 90$ のとき約 730 分の 1 となり 1 つあたり 3 分以下での半魔方陣の生成に成功した．また，世代数については $n = 60$ のとき約 86 分の 1，$n = 90$ のとき約 177 分の 1 となり世代数についても大幅な削減に成功している．さらに，生成時間・世代数ともに n が大きくなるにつれて改善度が大きくなっていることから次数の大きい魔方陣に対して提案手法の優位性が示せる．

なお，Xie & Kang の手法と局所修正部分のみに改善を加えた手法を比較したところ，改善を加えた先行研究+の手法では，生成時間は $n = 60$ のとき約 3 分の 1 となり，平均世代数が約 3 分の 2 へと削減ができている．

第 3 段階

表 3 に半魔方陣から魔方陣を生成するまでの各手法における実験結果を示す．Xie & Kang の手法 [4] では $n \ge 60$ 以上の魔方陣の現実的な時間での生成が困難となっているが，提案手法では $n = 100$ の魔方陣まで生成することができた．

また $n < 60$ の魔方陣の生成時間を比較すると，$n = 40$ では約 40 分の 1 となり生成時間も短縮することができている．本手法により生成された魔方陣および半魔方陣を GitHub[1] で公開する

5 考察

第 1・2 段階

局所修正の改善による性能改善度を測るため，Xie & Kang の先行研究 [4] と先行研究+の手法を比較したところ，先行研究+の方が生成時間が短かった．生成時間が短縮された要因としては局所修正の適用条件が緩和されたことが大きいと考えられる．2 列の和がそれぞれ定和になる交換条件より，いずれか一方の列の和が定和となる条件の方が満たしやすく，かつ，着実に 1 列ずつ揃っていくため効果が高いと考えられる．

1) https://github.com/takemi853/MS_3phase

表 2: 第 1・2 段階の平均生成時間および平均世代数 (10 trials)

	n	10	20	30	40	50	60	70	80	90	100
生成時間	Xie&Kang [4]	1.3(±0.5)s	18.0(±12.0)s	1.5(±0.5)m	6.4(±1.6)m	23.0(±4.7)m	1.1(±0.3)h	2.7(±1.1)h	6.0(±2.5)h	26.0(±11.7)h	≥50h
	先行研究+	0.4(±0.1)s	4.7(±1.5)s	32.0(±9.0)s	2.2(±0.9)m	6.9(±1.1)m	21.1(±4.2)m	44.0(±7.2)m	1.6(±0.6)h	3.2(±0.8)h	7.8(±4.8)h
	提案手法	0.04s	0.4(±0.1)s	2.0(±0.5)s	6.6(±2.5)s	16.8(±7.1)s	35.0(±17.4)s	1.3m(±1.1)	1.4(±0.4)m	2.2(±0.8)m	4.2(±1.4)m
世代数	Xie&Kang [4]	176(±38)	907(±213)	2213(±577)	5055(±911)	10560(±3065)	19832(±1759)	34780(±8152)	52570(±11706)	76844(±7148)	-
	先行研究+	72(±36)	322(±80)	1099(±247)	2701(±665)	5845(±779)	13096(±2082)	21674(±3902)	34750(±10805)	60254(±14234)	71634(±10739)
	提案手法	15(±5)	44(±6)	81(±13)	125(±20)	181(±9)	229(±28)	318(±45)	355(±33)	433(±36)	497(±56)

(h,m,s はそれぞれ 時間，分，秒を表す)

表 3: 第 3 段階 平均生成時間・平均世代数比較 (10 trials)

	n	10	20	30	40	50	60	70	80	100
生成時間	Xie&Kang [4]	10.9(±13.1)s	48.5(±38.1)s	29.6(±17.6)min	2.1(±1.3)h	19.8(±9.5)h	≥30h	N/A	N/A	N/A
	提案手法	2.9(±3.1)s	40.6(±51.3)s	2.1(±3.0)m	6.4(±7.7)m	8.7(±9.4)m	29.5(±18.0)m	51.5(±50.0)m	54.5(±50.2)m	5.7(±5.8)h
世代数	Xie&Kang [4]	720(±707)	7358(±2886)	23680(±11878)	34484(±14810)	55490(±15534)	-	-	-	-
	提案手法	728(±1002)	3176(±2068)	5197(±2280)	4674(±3142)	5036(±3414)	5759(±3956)	6443(±3917)	8066(±4708)	14538(±6972)

(h,m,s はそれぞれ 時間，分，秒を表す)

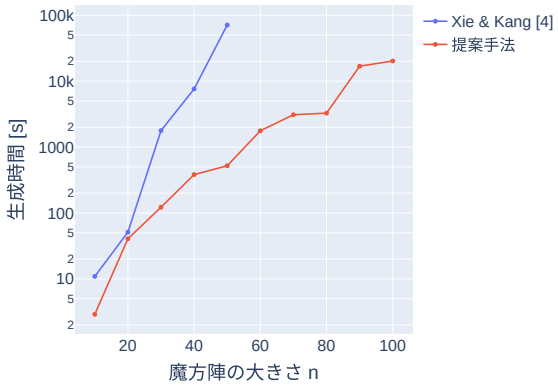

図 4: 第 3 段階の平均生成時間

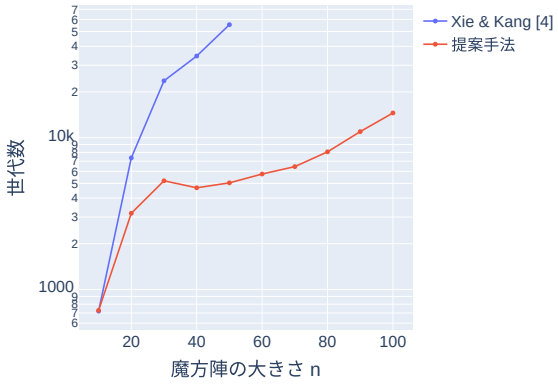

図 5: 第 3 段階の平均世代数

次に，提案手法について考察すると表 2 の結果から局所修正のみの改善である先行研究+よりも大きな削減ができている．これは，第 1 段階の追加による以下の 2 つの影響が大きいと考えられる．

第一の影響は，局所修正の時間短縮である．局所修正では，各行・列に対して条件を満たすマスを総当たりに近いアルゴリズムで探索するため，全体のアルゴリズムの中で最も時間を要する処理である．本研究では第 1 段階を追加し，各行の和が定和になるようにしたことで全ての行に対する探索が不要になったことから探索回数が大幅に減り，時間の短縮につながったと考えられる．

第二の影響は，世代数の削減である．局所修正条件の緩和と第 1 段階の追加により列方向のみの探索で十分となったことから，半魔方陣の生成に要する世代数が減少した．半魔方陣の生成時間は世代数に依存するため，生成時間の減少に大きく寄与していると考えられる．

第 3 段階

Xie & Kang の先行研究 [4] では，局所修正の適用と進化戦略の変更のための閾値を定数 100 としていた．魔方陣の次数が増加するにつれて適応度も大きくなるにもかかわらず，閾値が定数であるため閾値以下になる条件が厳しくなると考えられる．そのため，再現手法では $n \geq 60$ におけるほとんどの場合，この閾値よりも大きい値で局所解に陥っていたことから，時間をかけても魔方陣が生成できなかったのではないかと考えられる．そのため，提案手法では動的な閾値 $0.45n^2$ とすることで魔方陣の次数が増加するとともに，閾値も大きくなるようにした．その結果局所解に陥る場合が減少し，$n = 100$ の魔方陣生成につながったと考えられる．

6 おわりに

本稿では，ルールベースではない魔方陣の生成方法として，進化戦略を用いた手法に着目し，Xie & Kang の先行研究 [4] よりも短時間で魔方陣を生成できるアルゴリズムを提案した．実験の結果，大幅に生成時間が短縮できたことから 3 段階構築を用いた提案手法が魔方陣を効率よく生成できることが示せた．

第 1 段階を高速に実装するために本稿では偶数次の魔方陣のみについて取り扱ったが，今後の研究では奇数次の魔方陣生成へも第 1 段階を工夫することで応用する．

また，今回は一般的な魔方陣について効率の良い生成方法を考えたが，特定の条件を付与して作られる派生系の魔方陣である完全方陣や多重魔方陣，そして 3 次元に拡張した立方陣へも適用可能と考える．

参考文献

[1] Pickover, C. A.: The Zen of magic squares, circles, and stars, in *The Zen of Magic Squares, Circles, and Stars*, Princeton University Press (2011).

[2] M, K.: Magic Squares, in *Mathematical Recreations*, pp. 142–192, W.W.Norton & Company Inc, New York (1942).

[3] Kitajima, A. and Kikuchi, M.: Numerous but rare: An exploration of magic squares, *Plos one*, Vol. 10, No. 5, p. e0125062 (2015).

[4] Xie, T. and Kang, L.: An evolutionary algorithm for magic squares, in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, Vol. 2, pp. 906–913IEEE (2003).

[5] Bäck, T., Hoffmeister, F. and Schwefel, H.-P.: A survey of evolution strategies, in *Proceedings of the fourth international conference on genetic algorithms*, pp. 2–9Citeseer (1991).