推定収束点を用いた進化計算高速化の評価

裴岩 †，余俊 ††，高木英行 ‡

会津大学コンピュータ理工学部 †，
九州大学大学院芸術工学府 ††，
九州大学大学院芸術工学研究院 ‡

1　はじめに

進化計算研究の主要目的の一つは収束の高速化である．これまで色々が進化計算アルゴリズムの提案[2, 3, 7]，演算の改良[4, 5]，大まかな最適解への高速収束のためのfitness景観の近似[4, 5, 6]，等々の提案がなされて来た．

親世代探索点から子世代探索点への移動方向，あるいは，数世代を経た子孫世代探索点への進化パスを総称して以下では移動ベクトルと呼ぶことにしよう．この移動ベクトルは大局的最適解，あるいは，局所最適解に向かう情報を持っている（Fig. 1)参照）ので，これら移動ベクトルの方向情報を進化計算の収束高速化に利用可能である．

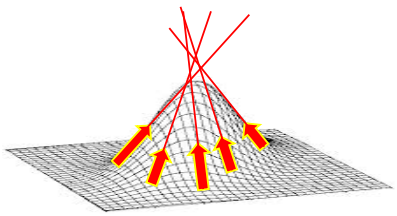


Fig. 1　親世代から子世代への移動ベクトルは，d次元探索空間上で大局的最適解近傍の1点に向かう．

筆者らは，親世代から子世代への複数の移動ベクトルが向かう収束点を数理的に求める方法を提案しベンチマーク関数で評価をした[8, 9]．この収束点は大局的最適解近傍であることが期待でき，この計算で求められて収束点をエリート個体として加えることで進化計算の高速化が期待できる．

しかしながら，この提案手法は原理的に単峰性のfitness景観には有効であっても多峰性の最適化タスクへの効果は保障されない．異なる収束点を目指す移動ベクトルを混ぜ合わせて得られた一つの収束点は，最適解近傍である保証がない．この基本提案手法を一般的な多峰性に適用できるよう拡張するには，移動ベクトルを収束方向でクラスタリングし，その後に基本提案手法することが解決法となる．そのための第一歩として，筆者らはd次元双極性タスク，すなわち，局最適解の他に一つの局所最適解を持つタスクに対処できるクラスタリング手法を提案した[11, 12]．

本論文の目的は，双極タスクに拡張した本提案手法で得られた収束点を進化計算の高速化に利用し評価することである．評価には，差分進化へ本手法の推定収束点を加えベンチマーク関数で差分進化の収束性能を評価する．

初めに移動ベクトルの収束点推定方法[8, 9]を簡単に第2.1節で紹介する．次に移動ベクトルをそれらの方向で二つにクラスタリングをする手法，および，それらのクラス毎の移動ベクトルの収束点精度を向上させる四つの改良法を，それぞれ第2.2節，第2.3節で紹介する．この評価実験と考察を第3節で述べる．

2　移動ベクトルの収束点推定とクラスタリングによる双極性タスクへの拡張

2.1　移動ベクトルの収束点推定

親世代の個体から次世代の子個体への移動ベクトルから，これら移動ベクトルが収束する点（複数の移動ベクトルに最も近い1点）は数学的に計算できる[8, 9]．この方法は次節のクラスタリグ後にそれぞれのクラス内の移動ベクトルの収束点推定に用いられる．

初めに記号の説明をする．Fig. 2の$\boldsymbol{a}_i$と$\boldsymbol{c}_i$はi番目の親個体とその子個体である（$\boldsymbol{a}_i, \boldsymbol{c}_i \in \mathbb{R}^d$）．

Evaluation of EC Acceleration by Using Estimated Points

† Yan Pei(peiyan@u-aizu.ac.jp)
†† Jun Yu(yujun@kyudai.jp)
‡ Hideyuki Takagi(takagi@design.kyushu-u.ac.jp)
School of Computer Science and Engineering, the University of Aizu (†)
Graduate School of Design, Kyushu University (††)
Faculty of Design, Kyushu University (‡)

次に，i番目の移動ベクトルは方向ベクトルとして定義される（$\boldsymbol{b}_i = \boldsymbol{c}_i - \boldsymbol{a}_i$）．$\boldsymbol{b}_i$の単位ベクトルを$\boldsymbol{b}_{0i} = \boldsymbol{b}_i/||\boldsymbol{b}_i||\boldsymbol{b}_{0i} = \boldsymbol{b}_i/||\boldsymbol{b}_i||$, i.e. $\boldsymbol{b}_{0i}^T\boldsymbol{b}_{0i} = 1$と定義する．すなわち，$\boldsymbol{b}_{0i}^T\boldsymbol{b}_{0i} = 1$である．

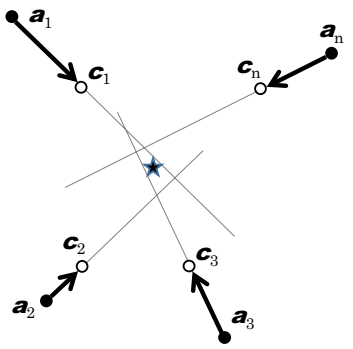


Fig. 2　移動ベクトル $\boldsymbol{b}_i$ $(= \boldsymbol{c}_i - \boldsymbol{a}_i)$ はd次元探索空間上の親個体$\boldsymbol{a}_i$とその子個体$\boldsymbol{c}_i$から計算できる．$\star$ 印はこれら移動ベクトルの収束点．

$\boldsymbol{x} \in \mathbb{R}^d$ をn本の拡張方向ベクトル$\boldsymbol{a}_i + t_i\boldsymbol{b}_i$ $(t_i \in R)$に最も近い点とする．この「近い」という意味は，$\boldsymbol{x}$からn本の拡張方向ベクトルへの最短距離の合計（式(1)のJ$(\boldsymbol{x}, \{t_i\})$）を最小にするという意味である．

収束点$\boldsymbol{x}$から拡張方向ベクトルへのへの最短線分は，$\boldsymbol{x}$からの直交写像なので，式(2)の直交条件を式(1)に代入しt_iを削除する．

$$\mathrm{J}(\boldsymbol{x}, \{t_i\}) = \sum_{i=1}^{n} \|\boldsymbol{a}_i + t_i\boldsymbol{b}_i - \boldsymbol{x}\|^2 \qquad (1)$$

$$\boldsymbol{b}_i^{\mathrm{T}}(\boldsymbol{a}_i + t_i\boldsymbol{b}_i - \boldsymbol{x}) = 0 \quad \text{（直交条件）} \qquad (2)$$

式(1)の距離合計を最小にする$\hat{\boldsymbol{x}}$は，$\boldsymbol{x}$の各要素で偏微分し0と置くことで求められる．最後に，収束点$\hat{\boldsymbol{x}}$は式(3)で与えられる．ここで，$\boldsymbol{I}_d$は単位行列である．

$$\hat{\boldsymbol{x}} = \left\{\sum_{i=1}^{n}\left(\boldsymbol{I}_d - \boldsymbol{b}_{0i}\boldsymbol{b}_{0i}^{\mathrm{T}}\right)\right\}^{-1}\left\{\sum_{i=1}^{n}\left(\boldsymbol{I}_d - \boldsymbol{b}_{0i}\boldsymbol{b}_{0i}^{\mathrm{T}}\right)\boldsymbol{a}_i\right\} \qquad (3)$$

プログラム言語によっては，逆行列演算を含む式(3)を計算することは手間である．そこで文献[8, 9]では，行列演算を含まないこの式の近似式，および，反復計算法 も提案しているので参照されたい．

2.2　双極対応の移動ベクトルのクラスタリング

第2.1節の移動ベクトル収束点推定法は単峰性タスクには確かに効果的である[8, 9]が，移動ベクトルが異なる局所最適解に向かう多峰性関数の場合は必ずしもそうではない．本手法を汎用化するには，まず，異なる局所解に向かう移動ベクトルをクラスタリングし，その後それぞれの局所解に向かう移動ベクトル毎にこの提案手法を適用する必要がある．この手法は新ニッチ手法になり得る．

この取組の第1ステップとして，まずは双極タスクを対象とし，移動ベクトルをクラスタリングする手法を開発し，二つの局所解を推定することから始める．このクラスタリング手法の6ステップをTable 1の6個の図で説明する[11, 12]．

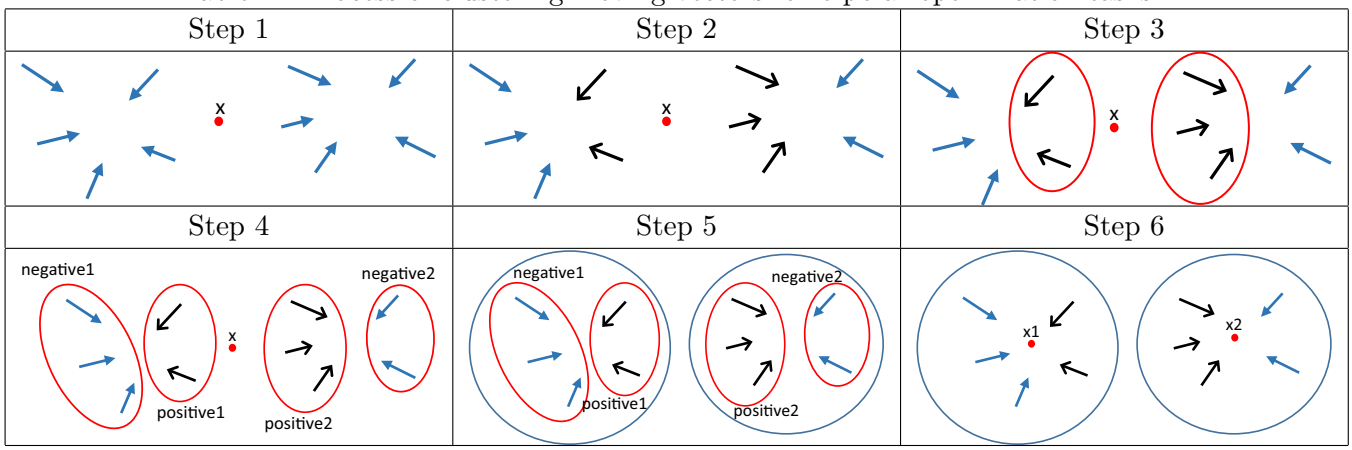
Step 1: 全移動ベクトルを使って，第2節の方法で収束点$\boldsymbol{x}$を求める．

Step 2: 得られた収束点$\boldsymbol{x}$に向かって来る移動ベクトルと離れていく移動ベクトルに二分する．判定には，移動ベクトルと推定収束点から移動ベクトル先端に向かうベクトルの内積の符号判定をすればよい．$\boldsymbol{b}_i^{\mathrm{T}} \cdot (\boldsymbol{c}_i - \boldsymbol{x})$が正であれば同じ向きを向いている（すなわち推定収束点から離れていく方向）し，負であれば移動ベクトルは推定収束点方向に向かっている．これら2クラスをpositiveクラスとnegativeクラスと呼ぼう．

Step 3: 全移動ベクトル終点の中で最遠にある2点を探し，$\boldsymbol{c}^1$と$\boldsymbol{c}^2$としよう．他のpositiveクラス内の移動ベクトルが$\boldsymbol{c}^1$と$\boldsymbol{c}^2$のどちらに向いているかでこれらpositiveクラスの移動ベクトルを二分する．判定には，$d1 = ||\boldsymbol{c}^1 - \boldsymbol{a}_i||/||\boldsymbol{c}^1 - \boldsymbol{c}_i||$と$d2 = ||\boldsymbol{c}^2 - \boldsymbol{a}_i||/||\boldsymbol{c}^2 - \boldsymbol{c}_i||$を用いる．$d1$が1より大きく$d2$が1より小さければ，$i$番目の移動ベクトル$\boldsymbol{b}_i$は$\boldsymbol{c}^1$方向を向いており，その逆であれば$\boldsymbol{c}^2$方向を向いている．$d1$, $d2$共に1より大きい，あるいは，小さい場合が生じた時は，$d1$と$d2$の大きい方に向いていると判断する．あるいは，そのような曖昧な方向の移動ベクトルは収束点計算に用いない，という選択肢もある．こうして，どちらの方向に向かっているかで，positiveクラス内の移動ベクトルを二分できる．

Step 4: negativeクラスの移動ベクトルはpositiveクラスの移動ベクトルよりも離れて二分できるので，クラスタリングアルゴリズムを適用する．例えば，最遠の移動ベクトル終

Table 1　Process of clustering moving vectors for bipolar optimization tasks.



点を2点探して初期値としてk-means++法[1]で2クラスに分ける．これらをnegative1サブクラスとnegative 2サブクラスと呼ぼう．

Step 5: Step 3とStep 4で四つのサブクラスができる．Step 3の$\boldsymbol{c}^1$と$\boldsymbol{c}^2$に向かっている移動ベクトル群と遠ざかっていく移動ベクトル群がStep 3で分かっており，さらに，Step 4で$\boldsymbol{c}^1$と$\boldsymbol{c}^2$がどのサブグループに属しているかが分かるので，四つのサブグループの中でペアになる二つのサブグループが分かる．Table 1では，(positive1とnegative1)，および，(positive2とnegative2)がペアになることが分かる．

Step 6: Step 5で同じ局所最適解に向かう移動ベクトル群が二つ得られたので，第2節の方法でそれぞれの収束点（$\boldsymbol{x}_1$と$\boldsymbol{x}_2$）を推定する．

2.3　提案クラスタリング手法の改善

進化計算の探索では，移動ベクトルが真っ直ぐ局所最適解に向かうことは保障できない．そのような移動ベクトルを含めて求める収束点は局所最適解から離れてしまう．さらに，前節のクラスタリングでも分類誤りが増え，同様に局所最適解の推定精度が悪くなる．

このために4種類の対策を提案した[11, 12]．

改良1 微小領域内で，ランダムに2個体を生成して移動ベクトル生成．

改良2 fitness上位個体のみの移動ベクトルを使ったクラスタリング，および，クラス毎の収束点の計算

改良3 直交ベクトルを用いた移動ベクトルの向きの補正．

改良4 クラス毎の収束点から同じクラス内の移動ベクトルの向きをチェックし，クラスタリング誤りを補正

3　評価実験と考察

提案手法で得られた収束推定点を使うことで進化計算を高速にできるかどうかを評価する．評価実験には差分進化を用いる．提案手法の移動ベクトル収束点をエリート個体として差分進化の最悪個体と置換する手法を「差分進化＋推定収束点」と表記することにしよう．DeJongの6関数（F1: Sphere関数，F2: Rosenbrock関数，F3: ステップ関数，F4: 単峰＋ノイズ，F5: Rastrigin関数，F6: Schwefel関数）をベンチマーク関数に用いて，通常の差分進化と「差分進化＋推定収束点」の比較評価を行う．

ベンチマーク関数は2次元と30次元で，全パラメータの探索範囲は，F1 が[-5.12, 5.12]，F2 が[-2.048, 2.048]，F3 が[-5.12, 5.12]，F4 が[-1.28, 1.28]，F5が[-5.12, 5.12]，F6が[-512, 512]である．これらの全最適化タスクは最適解が最小fitness値を持つ最適化問題で，連続性と非連続性，凸と非凸，単峰性と多峰性，低次元と高次元，という多様なfitness景観を持つ．

差分進化にはDE/rand/1/binを用い，100世代までの探索を30試行行い，各世代で差分進化 vs.（差分進化＋推定収束点）の平均fitnessに有意な差があるかどうかをWilcoxsonの符号検定で確認する．

Fig.s 3 と4は，各々2次元および30次元の最適fitnessの30試行平均の収束曲線である．2次元関

数の場合，危険率5%のWilcoxon符号検定でF3以外のすべてのベンチマーク関数で，通常差分進化と（差分進化＋推定収束点）に有意差が見られるが，全体的には大体同程度の収束性能とも言える．30次元関数の場合は，F1〜F5関数で，危険率5%のWilcoxon符号検定で通常差分進化と（差分進化＋推定収束点）に有意差が見られる．30次元関数F6の場合，収束点を一つだけ推定する基本法[8, 9]を適用すればFig. 4(f)のように従来法よりも悪くなるが，2つの最適解を推定する方法[11, 12]を適用すると従来法よりも収束性能が良くなる．

また2次元と30次元の移動ベクトル収束点と最適解へのユークリッド距離を各々Fig.s 5と6に示す．

Fig.s 3と4のWilcoxon符号検定結果付の収束曲線，および，移動ベクトルの推定収束点と最適解との距離から次のことが言える．

1. 2次元と30次元の両者で,（差分進化＋推定収束点）は通常差分進化より有意に高速化している．
2. 30次元の多峰関数（F5とF6）では,（差分進化＋二つの収束点推定法[11, 12]）を用いて得た推定点が（差分進化＋一つの収束点推定法[8, 9]を用いて得た推定点）より有意に差分進化を高速化している．
3. 30次元での高速化の効果は2次元の場合よりも大きい．
4. 二つの推定収束点を用いた提案推定点[11, 12]は，単峰関数の場合には一つの推定収束点を，多峰性関数の場合には二つの推定収束点を検出できることを確認した．
5. 収束点推定精度はタスクに依存する．

4 結論

親世代の探索点から子世代の探索点への移動ベクトルを推定する探索アルゴリズムの評価を行った．推定した最適解を差分進化の最悪個体と置き換えることで通常差分進化をより高速化することができることを評価実験で示した．

今後は双極タスクの最適解を推定する手法を一般的な多峰性の最適解推定ができるようアルゴリズムを改良し，多峰性での進化計算高速化に寄与することを示していく予定である．

謝辞

本研究はJSPS科学研究費（課題番号 15K00340, および，26540145）の助成を受けたものである．

参考文献

1) Arthur, D. and Vassilvitskii, S., “k-means++: the advantages of careful seeding,” 18th ACM-SIAM symposium on discrete algorithms (SODA2007), PA, USA, pp.1027–1035 (2007).

2) Back, T., Hammel, U., and Schwefel, H.-P., “Evolutionary computation: Comments on the history and current state,” IEEE Trans. on Evolutionary Computation, vol.1, no.1, pp.3–17 (1997).

3) Coello Coello, C.A. “Evolutionary multi-objective optimization: A historical view of the field,” IEEE Computational Intelligence Magazine (2006).

4) Das, S. and Suganthan, P.N. “Differential evolution: A survey of the state-of-the-art,” IEEE Trans. on Evolutionary Computation, vol.15, no.1, pp.4–31 (2011).

5) Eiben, Á.E., Hinterding, R., and Michalewicz, Z., “Parameter control in evolutionary algorithms,” IEEE Trans. on Evolutionary Computation, vol.3, no.2, pp.124–141 (1999).

6) Jin, Yaochu, “A Comprehensive survey of fitness approximation in evolutionary computation,” Soft Computing, Springer, vol.9, no.1, pp.3–12 (2005).

7) Mullen, R.J., Monekosso, D., Barman, S., and Remagnino, P., “A review of ant algorithms,” Expert Systems with Applications, vol.36, no.6, pp.9608–9617 (2009).

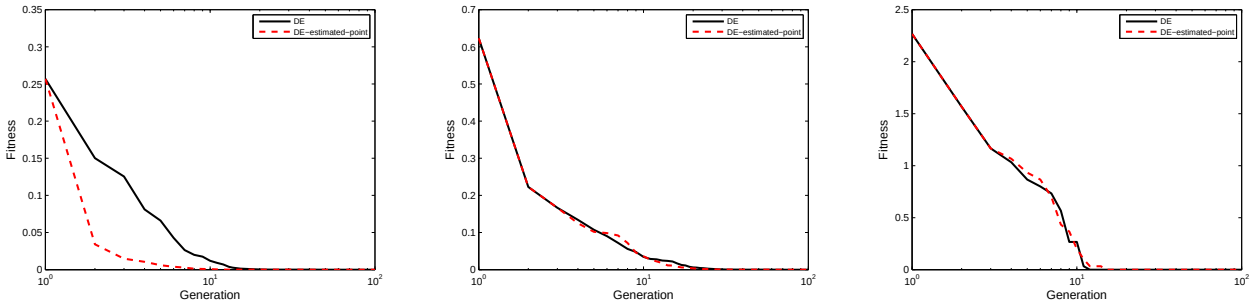
8) 村田昇，西井龍映，高木英行，裴岩「世代間移動ベクトル群の収束点推定法」2014進化計算シンポジウム，廿日市市, pp.341–350 (2014年12月20-21日).

9) Murata, N., Nishii, R., Takagi, H., and Y. Pei, “ Analytical Estimation of the Convergence Point of Populations,” 2015 IEEE Congress on Evolutionary Computation (CEC2015), Sendai, Japan, pp. 2619–2624 (May 25-28, 2015).

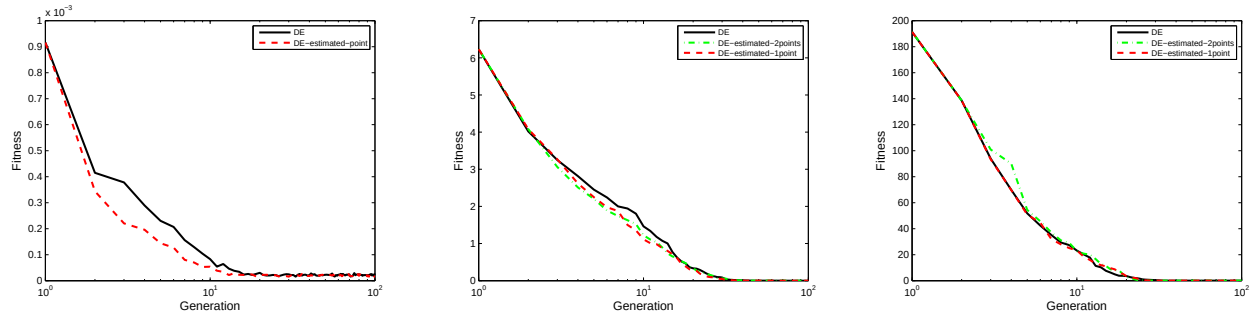
10) R. Storn and K. Price, “Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces,” *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

11) 余俊，高木英行「多峰性最適化問題での局所最適解推定高度化のための補正法 －局所最適解が2個の場合－」第7回進化計算研究会，神戸, pp.92–97（2015年9月7–8日）.

12) Yu J. and Takagi H., “ Clustering of Moving Vectors for Evolutionary Computation,” 7th International Conference on Soft Computing and Pattern Recognition (SoCPaR2015), Fukuoka, Japan, pp.169–174 (13-15, Oct. 2015).

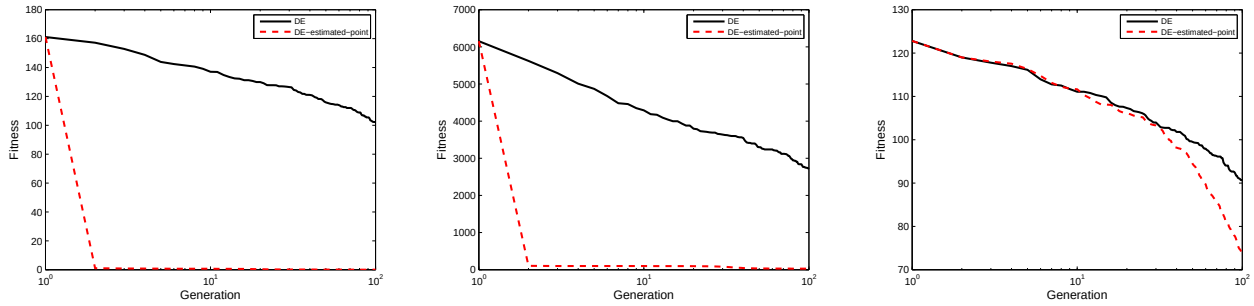


(a) 2次元ベンチマーク関数F1. (b) 2次元ベンチマーク関数F2. (c) 2次元ベンチマーク関数F3.

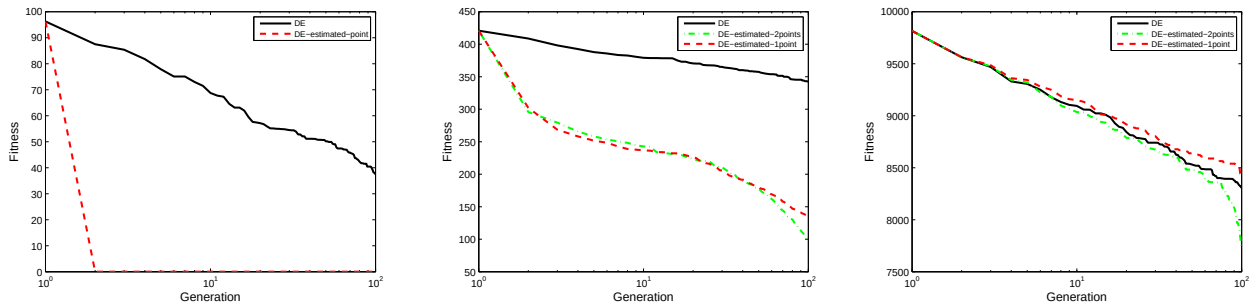


(d) 2次元ベンチマーク関数F4. (e) 2次元ベンチマーク関数F5. (f) 2次元ベンチマーク関数F6.

Fig. 3　2次元 DeJong関数の収束曲線.

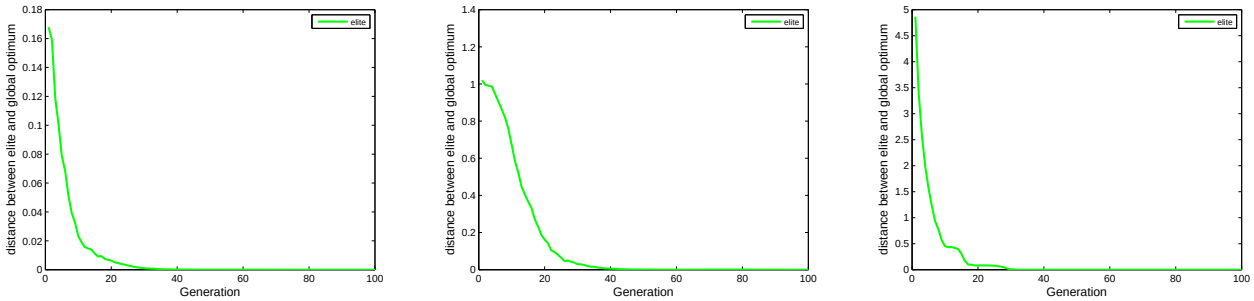


(a) 30次元ベンチマーク関数F1. (b) 30次元ベンチマーク関数F2. (c) 30次元ベンチマーク関数F3.

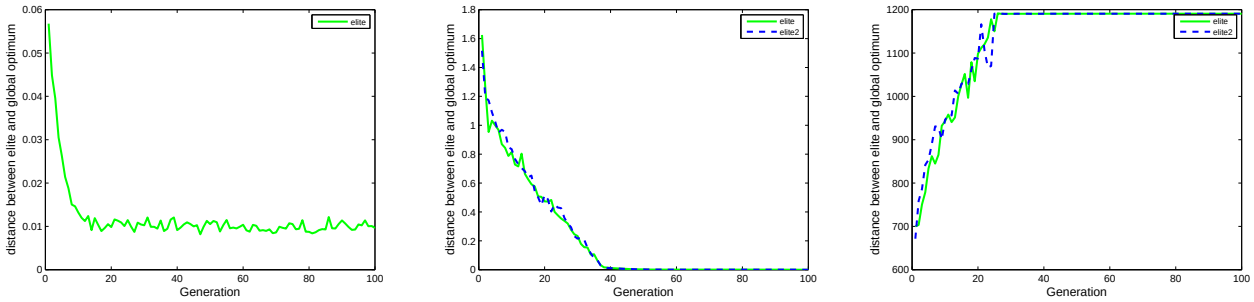


(d) 30次元ベンチマーク関数F4. (e) 30次元ベンチマーク関数F5. (f) 30次元ベンチマーク関数F6.

Fig. 4　30次元 DeJong関数の収束曲線.

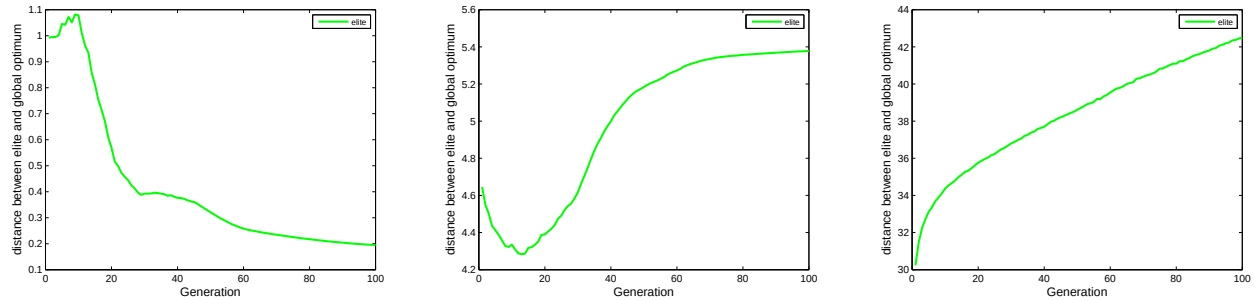


(a) 2次元ベンチマーク関数F1. (b) 2次元ベンチマーク関数F2. (c) 2次元ベンチマーク関数F3.

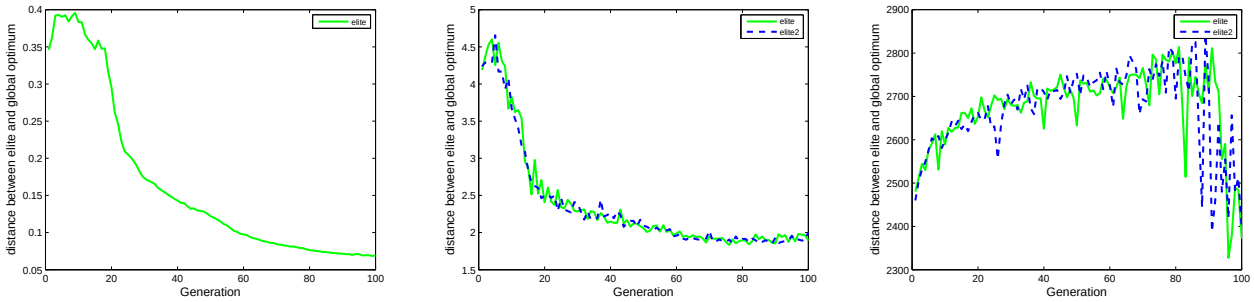


(d) 2次元ベンチマーク関数F4. (e) 2次元ベンチマーク関数F5. (f) 2次元ベンチマーク関数F6.

Fig. 5　2次元関数での収束推定点と局所最適解とのユークリッド距離.



(a) 30次元ベンチマーク関数F1. (b) 30次元ベンチマーク関数F2. (c) 30次元ベンチマーク関数F3.



(d) 30次元ベンチマーク関数F4. (e) 30次元ベンチマーク関数F5. (f) 30次元ベンチマーク関数F6.

Fig. 6　30次元関数での収束推定点と局所最適解とのユークリッド距離.