修士論文

「機械学習を用いた差分進化の性能改善」

2015 年2月 5 日提出

指導教員　　伊庭 斉志　教授

東京大学大学院　情報理工学系研究科
電子情報学専攻

学生証番号 48-126408　牛山 泰伸

目次

1 概要

DE(Differential Evolution)は進化的アルゴリズムの一種であり、最適化問題を解く際に用いられる手法である。速さ・ロバスト性に優れ、かつ使用が容易であることから、多くの最適化問題への応用が期待出来る。しかし、実用上の課題として、DE は解探索の際、適応度関数(fitness function)の評価回数が多くなるという欠点を抱えている. 評価1回あたりにかかるコストが大きいケースでは、高速性が失われてしまうため、関数評価回数を減らすことが求められてくる.

本論文では、この問題の解決するための手法として、DE の過程における変異（Mutation）時の個体選択に、機械学習（分類器）を導入することを提案する。これは、分類器による分類にて適切に個体を選択することで、DE における評価回数を減らすという方法である。この提案手法について説明し、各種性能実験を行い、そこから得られた課題について記す。

2 はじめに

ある関数（目的関数）が与えられた領域内で最小値（または最大値）をとるような解を求めるという問題を最適化問題と呼ぶ。簡単な例では、関数 $f(x)=(x-a)^2$ の最小値をとる x を求めるというものも、最適化問題の一種である。実世界においてもこのような問題は数多く存在するが、多くの場合は目的関数 $f(x)$ は複雑であり、解析的に解くのが難しく、何らかの解法が必要となってくる。

このような場合における解探索の有力な手法の一つに、進化的アルゴリズム(Evolutionary Algorithm, EA)というものがある。これは、生物の進化をモデルに作られた、最適化アルゴリズムの総称である。与えられた問題に対し、まずその解候補となるような個体（値、ベクトル）をランダムに複数作成する。次に、この作られた個体集団に対し、変異（Mutation）、交叉（Crossover）、選択（Selection）と呼ばれる各段階を踏んで、新しい（次世代の）個体集団を作成する。この変異・交叉・選択までの行程を 1 世代とし、これを何世代分も繰り返すことで、個体集団を世代を追うごとにどんどん「進化」させていき、個体を解に近づけていこうというものが、進化的アルゴリズムの考え方である。この進化的アルゴリズムには様々な種類の手法が存在する。例えば遺伝的アルゴリズム(GA)、粒子群最適化（PSO）、蟻コロニー最適化(ACO)、人工蜂コロニーアルゴリズム(ABC)などの手法が挙げられる。

本論文の研究対象である差分進化（Differential Evolution、以降 DE と表記）、上記の例と同様に進化的アルゴリズムの一種に分類される。DE の大きな特徴として、進化的アルゴリズムの中でも変異の段階の操作が簡単であり、とても扱いやすい手法であることが挙げられる。また高速性・ロバスト性も兼ね備えており、最適化問題を解く際の有用なアルゴリズムとして使用されている。例えば構造設計、電磁気学、回転翼の形の最適化に用いられているケースがある。

しかし、DE には他の多くの進化的アルゴリズムと同様、関数の評価回数が多くなってしまうという欠点もある。これは、毎世代 Mutation と Crossover のステップを経て作成される個体全てに対して、Selection の際に関数評価を行うという仕様のためである。すなわち、集団の総個体数 N、進化させる世代数を G とすると、N×G 回分の関数評価が必要ということになってしまう。

これは、1 回あたりの関数評価コスト（時間）が小さい場合、すなわち関数 $f(x)$ に Mutation・Crossover を経て作成された個体（ベクトル）x を代入したときにかかる計算量が小さい場合は問題にならない。しかし、 $f(x)$ が複雑化し、1 回あたりの評価コストが大きくなってしまうと、DE 全

体にかかる関数評価回数の多さの影響は無視出来なくなり、DE の解探索速度が落ち性能が低下してしまう。実世界では $f(x)$ の複雑化・x の多次元化などから、評価コストが大きくなることも多い。そのようなケースに DE を適応する場合、関数評価回数自体を削減するということは、評価コストによる性能低下の影響を抑えることが出来、DE の性能向上に繋がってくる。この考えから DE の性能改善に取り組んでいる先行研究として、個体選択に工夫をしている ILSDE、関数評価する際に分類器による選別を行う SVC-DE がある。

本研究はこれらの先行研究と同様、関数評価回数を減らすための新たな手法を提案する。
通常の DE において、Mutation 時の個体選択はランダムに行われている。この部分に機械学習による分類を適用する。Mutation 時の個体選択を良いものだけ行うことが出来れば、Mutation・Crossover の過程を経て作られる個体も良くなり、結果的に解収束速度が近づくと考えられる。この考えに基づき、Mutation 時の個体選択において、良い個体選択、つまり Mutation・Crossover を経て、解に近づく個体となると予想されるもの（進化すると予想されるもの）を正例とする。そうでない（進化しない）と予想される個体選択を負例と分類する。そして、正例の場合のみ Mutation・Crossover・Selection の処理を行うことで、関数評価の回数を減らすことが本手法の大きな狙いである。
本手法においては分類の精度の改善がそのまま性能改善に直結する。個体選択時の正例・負例分類の的中率が上昇すれば、それだけ解の収束に必要な関数評価回数は少なくなる。逆にうまく分類出来ない場合は、せっかくの機械学習導入が生きてこず、性能も上がってこない。このため、本提案手法では、分類精度を上げるため様々な工夫を試み、手法の性能上昇に取り組んでいる。

本論文の次節以降は次のようになっている。まずはじめに、DE について説明する。次に本研究の先行研究である、ILSDE および SVC-DE について紹介する。そして、本提案手法について紹介・説明を行う。提案手法の分類精度上昇のために行った各種実験およびその考察を述べる。そして先行研究と本提案手法との性能比較評価を行い、その結果について考察する。最後に、本研究のまとめと今後の課題について述べる。

3 差分進化

本章では、今回の研究対象となるアルゴリズム、差分進化(DE)について、その詳細を説明する。
DE は Storn らによって提案された手法[1]であり、進化的アルゴリズムの一種である。具体的な手法の流れは以下の通り。

Step1. Initialize（ベクトル集団の作成）
以降の計算のベースとなる D 次元ベクトル $\boldsymbol{x}_i$ を N 個作り（ i = 1, 2, ・・・, N ）、ベクトル集団を作成する. 各ベクトルの値は、与えられた領域の中からランダムに決定される。

Step2. Mutation（変異）
ベクトル $\boldsymbol{x}_i$ について、この変異ベクトルを作成するために、ベクトル集団 x の中から 3 つの個体をランダムに選択。これを r1、r2、r3 とする.（ただし $r1 \neq r2 \neq r3 \neq i$ ）その後、以下の式によって変異ベクトル $\boldsymbol{v}_i$ を作成する.

$$v_i = x_{r1} + F \cdot (x_{r2} - x_{r3}) \qquad (1)$$

ここで、F はユーザー側が値を定義する変数である. 一般的には[0, 1]の間で定義する.
上式を i = 1,2, ・・・, N の全てに適用し、変異ベクトル集団 v を作成する.

Step3. Crossover（交叉）

step2 で作成された変異ベクトル v_i および元のベクトル（親ベクトルと呼ぶ） x_i を、次の式によって交叉させる.

$$u_{j,i} = \begin{cases} v_{j,i} & (if \ \ rand \le CR) \\ x_{j,i} & otherwise \end{cases} \qquad (2)$$

ここで CR は交叉確率を表し、ユーザー側が[0, 1]の間で定義する. j はベクトルの次元を表す。この式は、ベクトルの各次元 j = 1, 2, ・・・, D において、確率 CR で u = v となり、それ以外は u = x になることを示している. すなわち、親ベクトルの各次元の値が、CR の確率で変異ベクトルのものに置き換わる、という式である。作成されたベクトル u_i を子ベクトルと呼ぶ。なお、u と x が完全に同じにならないことを保証するため、j のいずれか1つは必ず u = v となるようにする.
これを、i = 1, 2, ・・・, N に適用し、子ベクトル集団 u を作成する.

Step4. Selection（生存者選択）

子ベクトル u と親ベクトル x を適応度関数（＝目的関数）によって評価する. より最小値に近いベクトルを次世代の親ベクトルとして残す. 式としてあらわすと、下式のようになる.

$$next \ \ x_i = \begin{cases} u_i & if \ \ f(u_i) < f(x_i) \\ x_i & otherwise \end{cases} \qquad (3)$$

以降、新たに作られた次世代のベクトル集団 x について、同様に step2～4 の処理を行い、また次世代のベクトル集団を作成する。これを繰り返し、世代数を増やすことで、ベクトル集団 x を徐々に解に近づけていくアルゴリズムとなる。

この手法の大きな特徴は、Step2 の Mutation における、変異ベクトル u の作成方法である。式(1)の通り、DE においては、個体 r1 から(r2- r3)の差分をとることで、変異ベクトルを作成している。これは、ユーザー側が変異方法について、制御パラメータを F 以外用いる必要がなく、ベクトル個体集団から自動的に変異ベクトルを作成出来る式であることを示している。多くの進化的アルゴリズムにおいては、適切な変異をするためにユーザー側の制御が必要となるが、DE はその役目は差分ベクトルが行うため、扱いやすいアルゴリズムとなっている。

その他の step においても、簡易な式によって Crossover 等の各処理が行われていることが分かる。このため、最適化問題が与えられた際、この DE のアルゴリズムを実装することは非常に容易である。さらに、解探索の高速性や高いロバスト性も示されており、DE は最適化問題を解く際の有

力な手法であることが分かる。

しかし、式(3)からも分かるように、作成された子ベクトル全てにおいて比較のため $f(u_i)$ の値を求める必要があり、 $f(x)$ を計算する回数は多くなる。そのため $f(x)$ の評価(計算)コストが大きくなる場合は、DE 全体の性能低下に繋がってしまうという欠点もある。

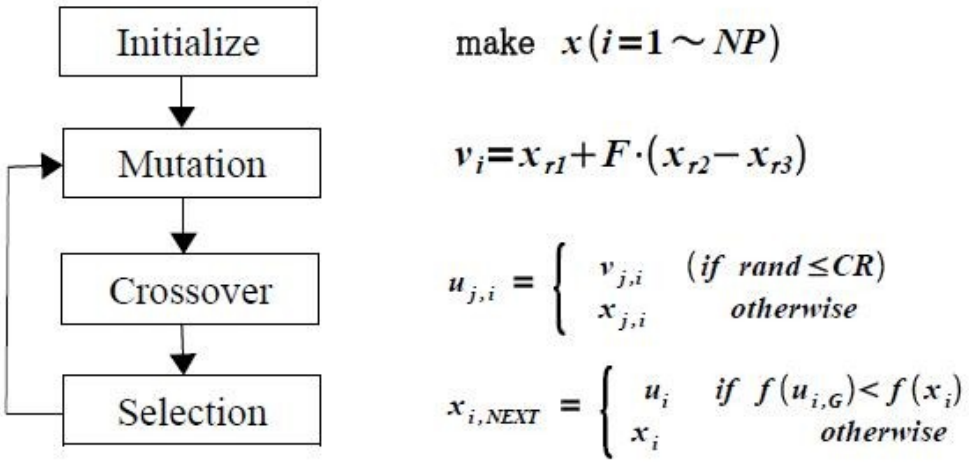


図 1. DE の流れ

```
Gmax: Generation size   N: Population size   D: Dimention size
f(x): Fitness function

 1: Initialize(x)
 2: for (g = 1 ~ Gmax)
 3:      for (i = 1 ~ N)
 4:              /*Mutation*/
 5:              select r1,r2,r3
 6:              v[i] = x[r1] + F * (x[r2] − x[r3])
 7:              /*Crossover*/
 8:              for(j = 1 ~ D)
 9:                      if(rand ≤ CR)
10:                             u[j][i] = v[j][i]
11:                     else
12:                             u[j][i] = x[j][i]
13:                     end
14:             end
15:             /*Selection*/
16:             if(f(u[i]) < f(x[i]))
17:                     x[i] = u[i]
18:             end
19:     end
20:end
```

図 2. DE のアルゴリズム

4 先行研究

本章では、DE の性能を改善する先行研究例として、ILSDE および SVC-DE について紹介する。

4.1 ILSDE

3 章の Step2 にて説明した通り、通常の DE では式(1)にて変異ベクトルを作成する際、ベクトル集団 x からランダムに r1, r2 , r3 をランダムに選択する. すなわち、子ベクトルの生成に大きな影響を及ぼす変異ベクトルの作成は、このランダムで選ばれた個体から作成される。この r1～r3 の 3 個の個体選択をランダム選択より良い方法で選択してくることが出来れば、より優れた変異ベクトルを作成することが出来、子ベクトルも良いものとなり結果的に解の収束速度が上がる（少ない評価回数で解を求められる）と考えられる。

ILSDE（DE based on Improved Learning Strategy）[2]はこの考え方から、Mutation 時の個体選択を工夫することで、DE の性能を向上させた手法である。具体的には、次のような戦略をから r1～r3 の 3 個の個体（ベクトル）を選択する。

まず、ベクトル集団 x から K 個ランダムに選択してくる（K は 3 以上の整数）。そして、K 個の中でもっとも適応度関数の評価値が良い、つまり $f(x)$ の値が最小となるようなベクトル x（best1）を r1、2 番目に良いベクトル（best2）を r2 とする。そして、最も評価値の悪い、つまり $f(x)$ の値が K 個中で最も大きくなるベクトル（worst）を r3 とするような個体選択方法である。

したがって、ILSDE において、DE の式(1)に相当する変異ベクトル作成式は、次式のように表すことが出来る。

$$v_i = best1 + F \cdot (best2 - worst) \qquad (4)$$

考え方としては、現時点で良い評価値を得ているベクトル（best1）に、悪いベクトル(worst)から良いベクトル（best2）の差分ベクトル（方向ベクトルの役割を果たす）を足していることになる。そのため、より良い評価値となる変異ベクトルの作成を期待出来る式になっている。

Crossover 以降の処理は通常の DE と同じであるため、ILSDE は DE における式(1)を式(4)と置き換えただけであり、手法としては単純である。しかしながら、これだけの変更にも関わらず、ILSDE は多くのベンチマーク関数において、通常の DE より性能が向上していることが実験結果から示されている。式(4)が良い個体選択によって良い変異ベクトルを作成出来ていることが結果からも言えること。また、この手法のように Mutation 時の r1～r3 の個体選択を何らかの戦略性をもって行うことは、DE の性能改善に繋がることが、本手法から伺うことが出来る。

```
Gmax: Generation size   N: Population size   D: Dimension size
f(x): Fitness function   kx: Select K num vector from x

 1: Initialize(x)
 2: for (g = 1 ~ Gmax)
 3:      for (i = 1 ~ N)
 4:              /*Mutation*/
 5:              select  K vector at random (= kx)
 6:              best1 = min(kx)
 7:              best2 = second min(kx)
 8:              worst = max(kx)
 9:              v[i] = best1 + F ∗ (best2 − worst)
10:
11:              DE-Crossover
12:
13:              DE-Selection
14:     end
15:end
```

図 3. ILSDE のアルゴリズム

4.2 SVC-DE

SVC-DE(Support Vector Classification – DE)[3]は、サポートベクターマシン(Support Vector Machine、SVM)[13]による分類を使用した手法である。

DE における関数評価は、式(3)から明らかなように Selection 時の親ベクトルと子ベクトルの比較のために行われる。子ベクトルが実際には親ベクトルより評価値が悪く、結果的に親ベクトルがそのまま次世代に残るようなケースでも、関数評価をする必要がある。ここで、もし子ベクトルの評価値が親より良くなるか否かを、関数評価前にあからじめ予測出来たとする。すると、そのケースでは関数評価を行わず親ベクトルを残すようにすることで、関数の評価回数を減らすことが出来る。

SVC-DE ではこの考え方に基づいた手法であり、子ベクトルの評価値予測を分類器（SVM）を使用して行っている。

アルゴリズムの大まかな流れは以下のようになる。なお、Mutation・Crossover の段階で行う処理は（学習例を得るための操作を除き）通常の DE と同じである。

1. まず通常の DE を一定世代分行い、ベクトル x およびその関数評価値 $f(x)$ を学習の訓練例として蓄積する。
2. Selection 時、子ベクトル u_i について、上記 1 によって蓄積された学習例の中から u_i と距離が近い学習例を順に k 個取り出す。
3. k 個の学習例について、正例・負例を決定する。子ベクトル u_i の親ベクトル x_i の評価値 $f(x_i)$ について、取り出した k 個の学習例の評価値と比較を行い、評価値が良くなるような例は正例、悪くなる例は負例とラベリングする。
4. 3 の方法でラベリングした k 個の学習例について、全て正例であるならば u_i は正例、全て負例ならば u_i は負例であると分類する。
   k 個の中に正例・負例ともに含まれる場合、SVM を使用して u_i の正例・負例の分類を行う。
5. u_i が正例と判定された場合、 u_i の関数評価値 $f(u_i)$ を計算し、親ベクトルの評価値との比較を行い、より良い方を次世代に残す。（通常の DE と同じ処理）またこの際、$(u_i, f(u_i))$ を新たな学習例として蓄積する。
   u_i が負例と判定された場合は何もせず、次世代には親ベクトル x_i がそのまま残る。

このように Selection の段階で分類器による機械学習を導入し、子ベクトルを評価対象とするか否か関数評価前に予測し、選別している点が、この手法の肝となる部分である。このようなやり方で、Selection 時の分類器による選別により関数評価回数を減らすことで、ILSDE 以上に DE の性能を改善することに成功している。

```
G: Generation size for collect , DB: data (x; f(x)) set
NB: k training data (x; f(x)) (neighbor u[i])

 1:    Initialize(x)
 2:    for (g = 1 ～ G)
 3:          do Mutation - Crossover - Selection
 4:          Archive all (xi; f(xi)) into DB
 5:    end
 6:    for (g = G ～)
 7:          do Mutation - Crossover
 8:          /*Selection*/
 9:          for each u[i]
10:                NB = Neighborhood(u[i], DB, k)
11:                case NB all positive
12:                      class = 1
13:                case NB all negative
14:                      class = -1
15:                case NB mixed
16:                      class = SVM(u[i], NB)
17:                end
18:                if class == 1
19:                      Archive all (u[i]; f(u[i])) into DB
20:                      if(f(u[i]) < f(x[i]))
21:                      x[i] = u[i]
22:                end
23:          end
24:    end
```

図 4. SVC-DE のアルゴリズム

5 提案手法

3 章で述べたとおり、通常の DE では変異ベクトルを作成する際、3 つの個体 r1～r3 をランダムに選択している。しかし、ILSDE の結果を見ても分かる通り、ただランダムで選択するよりも、より良い戦略を持って選択した方が良い結果が得られると考えられる。

本章ではこの考え方に基づき、また 4.2 節であった SVC-DE の手法も参考にし、Mutation 時の個体選択において、機械学習による分類を導入する手法を提案する。すなわち、分類器による分類にて、進化すると予想される個体選択（正例）のみ Crossover 以降の処理を行うようにする。進化しないと予想したもの（負例）について、Crossover 以降の処理は行わない。

5.1 アルゴリズムの概要

アルゴリズムの大まかな流れは以下の通り.

1. 通常 DE を一定世代分動かし、分類器用の学習例を収集する。
2. 収集後、Mutation 時の個体選択（r1, r2, r3）について分類器を使用し、正例か負例か分類する。
3. 正例と判断された個体選択の組のみ、Crossover 以降の処理に進める。負例と判断したものは、以降の過程の処理は行わない。
4. Selection 終了後、また 2 に戻る. これを一定の関数の評価回数になるまで繰り返す.

このように分類器を導入することで、作成される子ベクトルの評価値が悪くなるような、r1～r3 の個体選択を切り捨てることで、関数の評価回数の削減につなげようというアルゴリズムである。狙いとしては SVC-DE と似ているが、あちらと比べ、本手法では Mutation の段階から分類器を導入する点が大きく異なっている。単純に出来た子ベクトルの評価予測に分類器を使うのではなく、個体選択の段階から使うことで、より良い個体選択方法を作り出し、解の探索速度を上げ、さらなる性能改善につなげようというのが本提案手法の狙いである。

本手法で最も重要な点は、分類器の分類精度である。良い個体選択か否かの分類自体が上手くいかなければ、当然良い子ベクトルも作成出来ず、解の探索速度上昇にも繋がらない。いかに高い分類精度を持つよう工夫するかが、本研究における重要な課題となる。

分類精度を上げるためには、具体的には次の 3 点について考える必要がある。

- 分類手法
- 学習例の収集方法
- 特徴ベクトルのとり方

次節以降、上記 3 点について本提案手法が採用した方法を説明する。

5.2 学習例の収集方法

5.2.1 学習例の動的更新

本提案手法の分類では、あらかじめ学習例を収集し、それを元に分類する教師あり学習にて行う。分類は収集した学習例を元に行われるため、適切な学習例を用意することは非常に重要である。本手法では、最初の何世代分かは通常の DE を動かすことで、まずこの学習例を用意する。

しかし、これだけでは本手法の分類対象である、DE の個体選択の学習例としてうまく機能しない。通常であれば、一度学習例を用意すれば、その分類基準が変わることはないため、以降もその学習例を使用し続けることが出来る。しかし、DE の場合、世代が進むにつれベクトルが進化していく。順調に進化が進んだ場合、ベクトル集団は解にどんどん近づいていく。このため、以前の世代の基準ならば評価値が良いと言えたようなベクトルであっても、世代が進むにつれ評価値の基準もより解に近い、厳しいものとなるため、結果的に評価値が悪くなっていってしまう。以前の世代では正例と判断したケースでも、ベクトルが進化したために、負例と判断しなければならないケースが

出てくるのである。

そこで、学習例についても、ベクトル集団の進化に合わせて動的に更新していくことを考える。つまり、古い学習例は適宜捨てて、新しい学習例を積極的に分類時に使用していくようにする。このようにすることで、動的更新を適用しない場合に比べ、性能は大きく改善される。

図は bench1 f_1 において、学習例の動的更新なし（青線）と動的更新あり（橙線）による本提案手法の性能比較である。（分類手法は kNN をそのまま使用。特徴ベクトルは(r1,r2,r3)）動的更新なしの場合に比べ、動的更新を適用した場合は解の収束速度が速く、このケースでは同じ評価回数 1 万回で、更新なしの場合よりも 1 桁以上良い解を得ることが出来ている。

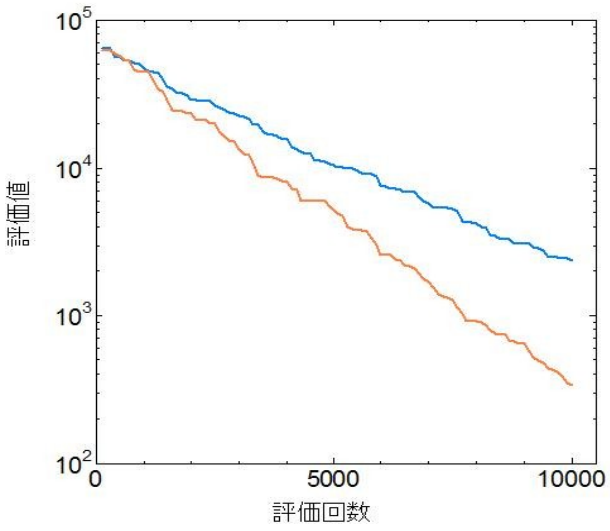


図 5.　学習例の動的更新の有無による性能変化
（青：動的更新なし　　橙：動的更新あり）

5.2.2 学習例の保存数

動的更新が効果的であることが前項から明らかになった。学習例を保存しておき、どんどん新しい学習例を蓄積していく。そして、一定数を超えたら、古い学習例を捨てていくという方式である。

ここで、どれだけの数の学習例を保存していくのが最も良いかという問題が出てくる。より最適な保存数を探るべく、いくつかのベンチマーク関数にて調査した結果が表 1 である。

表1　学習サンプルの保存個数による評価値の違い

	100 個	200 個	500 個	1000 個	2000 個
bench1−f_1	4.27E+02	3.46E+02	3.13E+02	3.37E+02	3.33E+02
bench1−f_{11}	5.47E+00	4.43E+00	4.04E+00	3.97E+00	3.85E+00

表より、学習例の保存数が少ない場合は、結果も悪くなることが分かる。100 個、200 個の時の評価値は他よりも劣っており、この個数では不十分であることが分かる。他方、500～2000 個の場合は評価値はほとんど変化ない。少ない場合は問題あるが、保存数は一定個数以上あれば、多い分にはあまり性能が変わらないことが分かる。

このように、学習例が多すぎても問題ない点については、実際にはベクトル集団が進化していくため、分類対象も進化し、古い学習例からは大きく離れた位置に分類対象が出来ること。また分類手法として後述するように kNN を使用している点が挙げられる。つまり、分類手法上、分類対象の周囲の学習例（つまり新しい学習例）しか使わないため、古い学習例は特に除去せずともノイズとはならないのである。

以上から学習例は新しい学習例をどんどん蓄積していくだけで良く、全て保存すれば問題ない。しかし、学習例の保存数が増えると kNN の分類をする際の計算コストも増えてしまう。そこで、本提案手法では表の結果から保存数は 1000 個あれば十分であると判断し、学習例の保存数は 1000 個と設定した。

5.2.3 学習例の偏り

本提案手法における分類は正例・負例の二値分類となる。このような二値分類では、学習例中の正例・負例は1:1の均等の割合であることが一般的には望ましい。しかし、本手法中に行う収集では、何もせずそのまま蓄積したのでは、このような均等な割合にはならないことが多い。以下の表に一部のベンチマーク関数における学習例 1000 個あたりの正例数および負例数を示す。（10 回学習例を収集し、その平均の値を示している。）

表2．正例・負例数

ベンチマーク関数	正例数	負例数	分散
bench1−f_1	96.1	903.9	127.69
bench1−f_5	38.6	961.4	92.84
bench1−f_9	93.3	906.7	109.41
bench1−f_{10}	106.5	893.5	49.65

上記のように、いずれのベンチマーク関数においても、正例:負例の比は1:9かそれよりさらに偏る結果となっている。負例の割合がこれだけ多くなってしまうのは、本分類の正例・負例の定義によるものである。今回正例と判定するものは、進化する(と予想出来る)ような個体選択のみなので、正例の数は簡単に進化出来るような例を除き、基本的には少なくなってしまうと考えられる。

このように正例・負例が偏ってる場合、分類が正常にいかない場合もあるため、学習例に何らかの補正を加えることが望ましいと考えられる。しかし、この学習例の偏りを緩和するような補正方法をいくつか試したものの、思ったほどの成果は得られなかった。変わりに、分類手法にてこのような偏りがあることを念頭に置き、改善を施したところ、そちらの方がよい成果を得られた。そのため、今回は学習例収集において、この学習例の偏りに対する補正処理は特に行っていない。

5.3 分類手法

5.3.1 k 近傍法

本提案手法では、分類手法として、分類器の中でも単純なアルゴリズムである k 近傍法(k-Nearest Neighbor, kNN)を用いる。ただし、さらに性能を改善するために、そのままの kNN ではなく、一部工夫したものを使用している。

k 近傍法とは、ある分類したい物について、それと(特徴空間上の)ユークリッド距離が近いような学習例を順に k 個とり、その学習例のクラスの多数決によって、分類対象のクラスを決定する手法である。今回は正例・負例の二値分類であるので、k 個中、正例の数の方が多ければ分類対象は正例、負例の方が多ければ負例という分類となる。例えば k=5 の場合、正例数=3、負例数=2 のケースでは正例数＞負例数なので正例、正例数＝1、負例数=4 のケースでは正例数＜負例数なので負例といったように分類される。

5.3.2 k 近傍法の改善

以上のような単純なアルゴリズムでありがら良い分類精度を示すのが k 近傍法の特徴である。しかし、本手法に適用する場合、このアルゴリズムそのままでは上手くいかなかった。単純な多数決による決定では、本来負例と判定すべきものを正例と判定するケースおよびその逆のケースが発生した。これは、5.2.3 項で述べたような、本手法の学習例の正例・負例の数の偏りが存在することが原因であると考えられる。また、全体的に正例・負例判定が「甘い」状態であった。

そこで、偏りを考慮し、かつなるべく厳しめに判定するよう、次のように kNN の手法を一部変更した。

まず，1 世代あたりで正例と判断するような個体選択数をあらかじめ決めておく(pselect) そして，より正例に近いと分類されるような個体を上から順に pselect 個のみ選択し，残りは負例と判断する

ようにした. より正例に近いような個体とは, ここでは学習例 k 個に含まれる正例の数がより多いものとした. つまり, 上の操作は言いかえれば, 各個体についてその近傍の学習例 k 個中に含まれる正例数が多いものから順に pselect 個分選び出し, これらの個体を正例と判断するということである. 例えば, k = 5 として, 集団のベクトル数が 50, pselect = 10 とする. この設定で各ベクトルについて近傍の学習例 k 個中, 正例 5 個のものが 3 個, 正例 4 個が 5 個, 正例 3 個が 10 個残りは正例 2 個以下だったとする. このケースでは, 正例 5 個のもの(3 個), 正例 4 個のもの(5 個), 正例 3 個のもの(2 個) の計 10 個が正例として判断されることになる. 正例 3 個のものは上記例では 10 個あるがこのように pselect を超過する場合は, 残り必要数である 2 個を正例 3 個であるベクトル計 10 個からランダムに選択する.

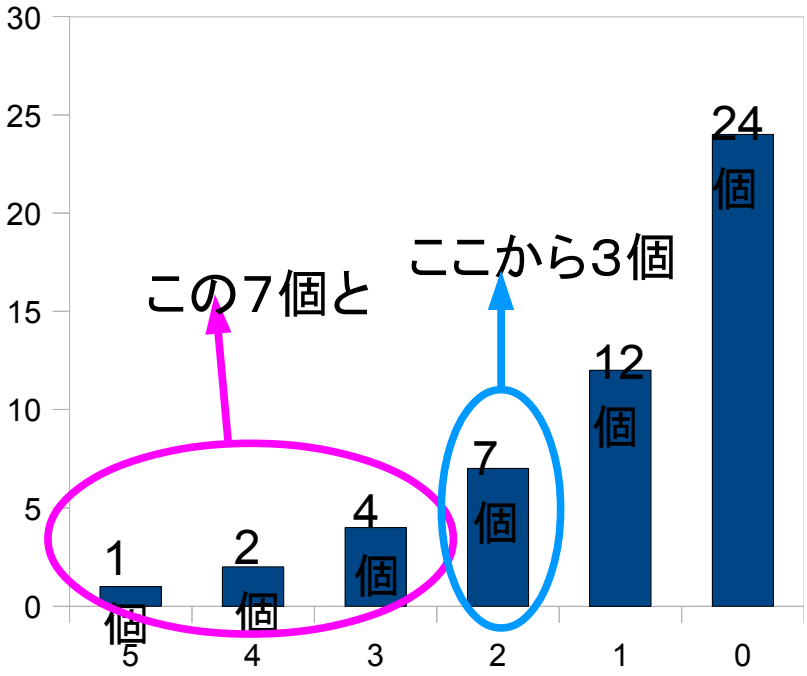


図 6　k=5、pselect = 10 の時の正例のとり方の例
（横軸：k(=5)個中の正例の個数、縦軸：該当するサンプル個数）

この手法にて重要なパラメータ pselect は、以下の式で決定している。

$$p\_select = (N * Positive / TrainSize) \times \beta \qquad (5)$$

ここで、N はベクトル集団数（＝1 世代で選択される学習サンプル数）、Positive は学習サンプル中の正例数、Trainsize は学習サンプル総数である。

これは、学習例の偏りに沿ったような選択をすることを狙った設定である。Positive／Trainsize は、サンプル数における正例の割合に他ならない。ここに N をかけるということは、1 世代分の個体選択総数のうち、学習例中の正例の割合と同じだけの割合だけ正例判定をしよう、ということである。

正例:負例＝1:9のような偏った学習例においても、1 世代あたりの正例判定をN の 10 分の 1 にすることで、その偏りと同様の結果となるように設定している。
ただし、それでは若干分類精度が低いことが分かっている。そこで、さらにこの設定に補正 β をかけることにした。この β の値設定については、実験を行った結果、 β=0.5 の時に良い性能を示した。これは、より正例に近いものを選択することが理想であるため、pselect の値を厳しく設定する(つまりより小さい値にする)ことになったことが、性能上昇に繋がったと考えられる。

この β は、解く対象となる関数によって最適な値が変化する可能性がある。そのため、 β は精度を上げるためにより深い検討を要するパラメータであると考えられる。 しかし、関数毎に β の値を変動させるのは、DE の特徴の一つである制御のしやすさ・簡易性を失うことになる可能性もある。
本研究では、簡易性を重視し、経験則から得た β=0.5 を全ての関数に一律に設定し、以降の実験を行った。

5.3.3 改善した kNN の性能評価

kNN を通常のものから、5.3.2 項で紹介した方法に変更することで、どの程度性能が変わったかいくつかのベンチマーク関数を使用して性能を評価した。なお、この時、他の分類手法として、SVC-DE が使用している SVM についても使用し、性能比較を行った。

手法 1:通常の kNN を使用したもの

手法 2:kNN に pselect を導入したもの(5.3.2 項で述べた改善を施した kNN)

手法 3:SVM(SVM-Light[17]を使用)

なお SVM については、学習例がそのままでは学習例の偏りのためほとんど機能しなかった。そこで、正例の平均をとり、その距離に近い順に全サンプル数の半分を正例と判定する、正例・負例を同数とするようなサンプル補正を行った。結果の一例は図 7。

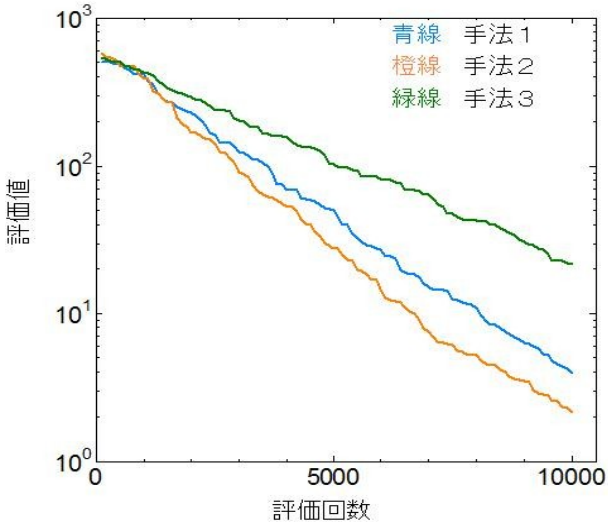


図 7. 分類手法の性能評価

結果から、性能の良い順に手法2(改善 kNN)、手法1(通常 kNN)、手法3(SVM)の順となった. 改善した kNN によって DE の性能が上がることがこの結果から確認された.

なお、SVM については、改善 kNN に劣るどころか、通常の kNN よりも悪くなるという結果となった. SVM 自体は kNN よりも精度の高い分類手法として知られる. にもかかわらず、本実験においては kNN に劣る結果となってしまった. これについては、以下の理由が考えられる.

本手法では個体がどんどん進化していくため、古い学習サンプルが使えなくなることは 5.2 節で述べたとおりである. SVM は、保存している学習サンプル全てを使って分類を行う手法である. そのため、もう使えないような古い学習サンプルが悪影響を与えてしまい、結果として精度が大きく落ちてしまったのではないかと考えられる. 今回のように、分類基準がどんどん変化していくケースでは、SVM のようにグローバルに判断する手法は適さない. それより、kNN のように自分の近くというローカルな部分のみから判断した方が良い結果を得られることが、本実験結果から推測できる.

5.4 特徴ベクトル

5.4.1 特徴ベクトル

一般的な分類器学習では、分類するデータは何らかの規則によって特徴ベクトルに変換し、データを特徴空間に 1 対 1 の写像で落とし込む。そして、その特徴空間上の位置・距離などからクラスの分類を行う。

このため、どのようなやり方で、データから特徴ベクトルを作成するか、すなわち特徴ベクトルの定義をいかに適切なものに設定するかは、分類精度に大きくかかわってくる問題である。適切な

特徴ベクトル設定をしなければ、それだけで分類精度が落ちてしまうため、どのような特徴ベクトルとするかは、一般的な分類においても工夫する余地がある所である。

5.4.2 本手法における特徴ベクトル

本手法の場合、Mutation 中の個体選択を分類対象としている。この個体選択は、r1～r3 の 3 つのベクトルを選ぶ必要がある。それぞれ別々に分類対象とすることも考えたが、それぞれは独立したものではなく最終的に作る変異ベクトルは 1 つであることから、r1～r3 の 3 つのベクトルを 1 つにまとめてしまうことが適切であると判断した。

したがって、今回はこの r1～r3、3 つのベクトルを何らかの方法で 1 つの特徴ベクトルにまとめる必要がある。また、分類対象は r1～r3 であるが、あくまで分類目的は優秀な個体選択を行うこと、つまり親ベクトルより良い変異ベクトルを作成することである。このため、r1～r3 だけでなくそれらのデータを用いて特徴ベクトルを作成することが有用である可能性がある。特徴ベクトルについては考えられる候補は複数あるため、どの方法が最適かしっかり検討を行う必要があると言える。

5.4.3 特徴ベクトルの性能比較

前項で述べたとおり、本分類において有用と考えられるデータは以下の通りである。

- 選択するベクトル　r1, r2, r3
- 変異ベクトル　v
- 親ベクトル　x

よって、これらのうちの一つを使用、あるいは複数を組み合わせて使用する考えがある。本研究では、以下の 5 種類の特徴ベクトル（fv1～fv5 と表記する）が有用ではないかと予想し、各々の場合の性能比較を行った。

1. **fv1 =(r1,r2,r3)**
   選択する個体 r1～r3 をつなげて，特徴ベクトルとしたものである.
   ここでいう「つなげる」とは、D 次元ベクトル r1 と同じく D 次元のベクトル r2 の場合では 1～D 次元までは r1 ベクトルであり、次の D+1～2D 次元までは r2 ベクトルとなる、ということである。
2. **fv2 = v**
   変異ベクトル v をそのまま特徴ベクトルとしたものである.
3. **fv3 = (v, x)**
   変異ベクトル v と親ベクトル x をつなげて特徴ベクトルとしたものである.
4. **fv4 = (r1, r2 − r3)**
   個体 r1 と，差分ベクトル(r2 − r3) をつなげて特徴ベクトルとしたものである.
   DE の Mutation で個体 r1～r3 がどのように使われるか考えた際，それぞれが等価に使われるのではなく式(1) にあるように個体 r1 とを差分ベクトル(r2 − r3) として使われることから，このようなとり方が優れていると予想した.

5. **fv5 = (r1, r2 − r3, x)**
   fv4 のベクトルに，さらに親ベクトルを追加した特徴ベクトルである．

以上 5 つの特徴ベクトルについて、ベンチマーク関数の一部で性能比較を行った。それぞれ 10 回行った平均値の結果が表 3 である。太字で示したものが、そのベンチマーク関数中で最も良い評価値であることを示している（太字が 2 つあるものは、t 検定において 5%の有意水準で両者に有意差が見られなかったものである）

表 3　特徴ベクトルの性能比較

	fv1	fv2	fv3	fv4	fv5
bench2− f_1	2.16E+02	**1.85E+01**	1.42E+03	3.83E+01	4.33E+02
bench2− f_4	4.41E+04	**3.09E+04**	7.73E+04	**3.08E+04**	5.13E+04
bench2− f_6	2.49E+06	**2.11E+05**	3.76E+07	7.72E+05	8.32E+06
bench2− f_9	2.32E+02	**2.17E+02**	2.57E+02	**2.06E+02**	2.32E+02

結果より、どの関数においても、おおよそ fv2、fv4、fv1、fv5、fv3 の順となった。すなわち、単純に特異ベクトル v をそのまま特徴ベクトルとしてしまった場合が最も良い結果を示し、次点で r1 と差分ベクトル(r2 – r3)をつなげた特徴ベクトルが続く。親ベクトル x の情報を付加した fv3、fv5 については、予想に反しかえって悪い結果となってしまった。

親ベクトルの情報を加えた場合に性能が落ちた件については、以下のように考察出来る。1 つのベクトルにだけ注目すれば、その親ベクトルと子ベクトルの比較によって進化の有無が決定される。しかし、DE は 1 つのベクトルだけでなくベクトル集団全体が徐々に解に近づいていく、つまり進化していく手法である。作られるべき子ベクトルは、その親ベクトルより良いことがもちろんだが、ベクトル集団全体から見た場合は、その集団全体の中でも良い子ベクトルが生成されることが望ましいと考えられる。対象となる親ベクトルが集団全体から見たら悪い値の場合、そのような親ベクトルより良くなっても、全体からは依然悪いままであるようなケースが発生しうるのである。言い換えれば、親ベクトルの情報を加えることは、かえって分類基準を「甘く」してしまうことになっているため、加えないほうが望ましいことが分かった。

特異ベクトルをそのまま直接特徴ベクトルとしたものが、最も良い結果となっている点については、特異ベクトルはそのまま子ベクトルにつながるため、これを分類時に見ることは影響が大きいということが言えてくる。特異ベクトルを単純な特徴ベクトルとしてしまう点については、まだ考察の余地があり、より良い特徴ベクトルの作り方がある可能性はある。しかし本研究においては、性能比較が示すとおり、この特徴ベクトル fv2 が最も良い性能を示した。

したがって、本提案手法において、特徴ベクトルは fv2、すなわち特異ベクトル v をそのまま特徴ベクトルとしたものを用いるようにする。

5.5 提案手法のまとめ

以上、5.2～5.4 節にて、分類精度を上げるために重要な学習例の収集方法、分類手法、特徴ベクトルのとり方について、様々な手法にて検討を重ね性能改善を行ったことを示した。以下、各方法について、本提案手法にて実際に採用する方法をまとめる。

- 学習例の収集方法
  最初に DE を一定世代分動かし学習例を集める。その後も結果を随時蓄積していき、学習例の動的更新を行う。学習例が 1000 個たまった時点で、古い学習例は順に捨てていく。

- 分類手法
  kNN を一部工夫したものを使用。1 世代あたり、より正例に近いようなものを、式(5)で定義した pselect 個分、正例と分類する

- 特徴ベクトルのとり方
  特異ベクトル u をそのまま特徴ベクトルとして使用する

したがって、本手法の具体的な流れは以下のようになる。

1. 通常 DE を一定世代分動かし、分類器用の学習例を収集する。
2. ベクトル x_i ($i=1\cdots N$)について、それぞれで個体選択(r1, r2, r3)をランダムで選ぶ
3. r1～r3 から出来る変異ベクトル u_i を、この個体選択における特徴ベクトルとする
4. N 個の特徴ベクトルについて、それぞれ近傍 k 個の学習例を取り出し、その正例数をカウント（0～k 個）
5. N 個の特徴ベクトルの中から、より正例に近い（学習例の正例カウント数が多い）順に pselect 個選択、これを正例と分類する。残りは負例と分類。
6. 5 にて正例と分類された個体選択(r1,r2,r3)の組のみ、Crossover 以降の処理に進む。負例のものは以降のステップに進まず、親ベクトルがそのまま次世代に残る
7. Crossover 以降の処理に進んだものについては、特徴ベクトルと Selection 時の結果から得たクラス（進化すれば正例、しなければ負礼）を、新たに学習例に追加していく。
8. 2～7 までを 1 世代分の処理とし、以降関数評価回数が一定値になるまでこれを繰り返す。学習例は 1000 個を超えた時点で、古いものから削除していく。

```
G: Generation size for collect , DB: data (x; f(x)) set
Emax: Max evaluation count

 1:    Initialize(x)
 2:    /*Collect Train Data*/
 3:     for (g = 1 ~ G)
 4:          for (i = 1 ~ N)
 5:               select r1,r2,r3
 6:               v[i] = r1 + F * (r2 − r3)
 7:               fv = v[i]          /*fv: feature vector*/
 8:               do Crossover
 9:               x[i] = Selection(x[i]; u[i])
10:               if(selected u[i])
11:                    class = 1
12:               else
13:                    class = -1
14:               end
15:               Archive (fv,class) into DB
16:          end
17:    end
18:    /*DE use classifier*/
19:    while (evaluation count < Emax)
20:          for(i = 1 ~ N)
21:               select r1,r2,r3
22:               fv[i] = v[i]
23:               pc[i] = k-neighbor(fv[i], DB)   /* pc: positive count*/
24:          end
25:          sort fv  (by positive count, descending order)
26:          cp = pop fv   (at pselect times)    /*cp:classify positive*/
27:          only cp case
28:          do  Mutation – Crossover – Selection
29:          if (selected u[i])
30:               class = 1
31:          else
32:               class = -1
33:          end
34:          Archive (cp, class) into DB
35:    end
```

図 8. 提案手法のアルゴリズム

6 提案手法の性能評価実験

計24個のベンチマーク関数に対して提案手法および他手法との性能比較を行い、提案手法の性能を評価した.

6.1 実験の設定

6.1.1 使用手法

(1)DE

通常の DE(3 章参照)

(2)ILSDE

先行研究(4.1 節参照)

(3)旧手法（工夫前の手法）

5.2～5.4 節にて行った性能の改善度合いを見るために計測

基本的な流れは提案手法と同様だが、以下 3 点について、下記のように現在の性能となる前に採用していた手法をとった

- 分類手法:通常の kNN
- 学習サンプル:最初に決定後は固定
- 特徴ベクトル:(r1, r2, r3)

(4)提案手法

5 章で説明したもの。旧手法との違いは以下

- 分類手法:kNN に一部工夫(5.3 節)
- 学習サンプル:動的更新 (5.2 節)
- 特徴ベクトル:v(変異ベクトル　5.4 節)

6.1.2 各種パラメータ設定

実験全体の設定

- 関数評価回数　10000
- 実験回数　10

DE の設定（ILSDE、提案手法も同様）

- ベクトル集団数 N = 100
- 差分の重みつけ F = 0.8　　（式(1)で使用）
- 交叉率 CR = 0.9　　（式(2)で使用）

ILSDE の設定

- ILSDE の K = 3　　（式(4)で使用）

提案手法の設定

- kNN の k=5
- 学習サンプルの保存数 TRAIN_SIZE = 1000
- 学習サンプル収集のために普通に動かす世代数 TRAIN_G = 10

6.1.3 ベンチマーク関数

性能評価を行うためのベンチマーク関数は2セット用意した.

1つ目のセットは論文[]にある f_1 ～ f_{15} までの 15 個の関数を使用した. 2つ目のセットは論文[]にある f_5 を除く f_1 ～ f_{10} までの9個の関数を使用した.

以降、1つ目のセットを bench1、2つめのセットを bench2 と表記する. 表2、表3に bench1、bench2 の各関数を記す. 表3中の式において、bias は大域解の値をずらすための定数、o はシフトする値、M は回転のための回転行列である. より詳細については論文[4]、[5]を参照.

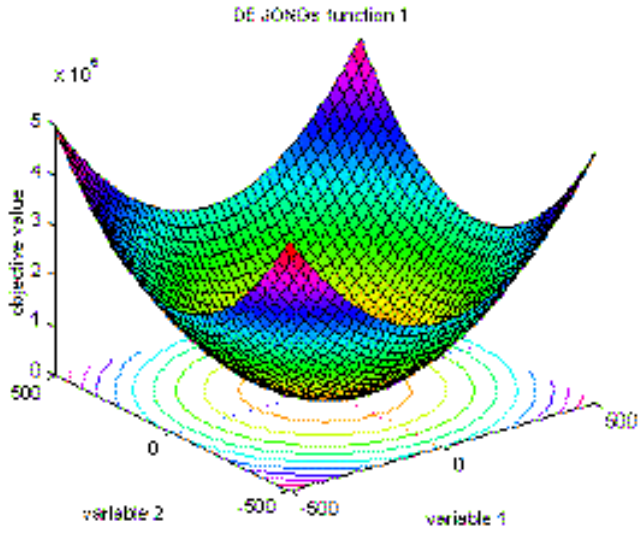
表 4.　ベンチマーク関数:bench1

	関数名	関数式	初期値の範囲
f_1	Sphere	$\sum x_i^2$	$[-100,100]^{30}$
f_2	Schwefel's Problem 2.22	$\sum \lvert x_i \rvert + \prod \lvert x_i \rvert$	$[-10,10]^{30}$
f_3	Schwefel's Problem 1.2	$\sum_i (\sum_j x_j)^2$	$[-100,100]^{30}$
f_4	Schwefel's Problem 2.21	$max_i(\lvert x_i \rvert, 1 \le i \le n)$	$[-100,100]^{30}$
f_5	Rosenbrock	$\sum [100(x_{i+1}-x_i^2)^2+(x_i-1)^2]$	$[-30,30]^{30}$
f_6	Step	$\sum (floor(x_i+0.5))^2$	$[-100,100]^{30}$
f_7	Quartic (+ Noise)	$\sum ix_i^4 + random[0,1]$	$[-1.28,1.28]^{30}$
f_8	Schwefel's Problem 2.26	$-\sum (x_i \sin(\sqrt{\lvert x_i \rvert}))$	$[-500,500]^{30}$
f_9	Rastrigin	$\sum [x_i^2 - 10\cos(2\pi x_i) + 10]$	$[-5.12,5.12]^{30}$
f_{10}	Ackley	$-20\exp(-0.2\sqrt{\frac{1}{n}\sum x_i^2}) - \exp(\frac{1}{n}\sum \cos 2\pi x_i) + 20 + e$	$[-32,32]^{30}$
f_{11}	Griewank	$\frac{1}{4000}\sum x_i^2 - \prod \cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-600,600]^{30}$
f_{12}	Penalized(1)	$\frac{\pi}{30}\{10\sin^2(\pi y_1) + \sum (y_i-1)^2[1+10\sin^2(\pi y_{i+1})] + (y_n-1)^2\} + \sum u(x_i,10,100,4)$	$[-50,50]^{30}$
f_{13}	Penalized(2)	$0.1\{\sin^2(3\pi x_1) + \sum (x_i-1)^2[1+\sin^2(3\pi x_{i+1})] + (x_n-1)^2[1+\sin^2(2\pi x_{30})]\} + \sum u(x_i,5,100,4)$	$[-50,50]^{30}$
f_{14}	Shekel's Foxholes	$\left[\frac{1}{500} + \sum \frac{1}{j+\sum (x_i-a_{ij})^6}\right]^{-1}$	$[-65.536, 65.536]^2$
f_{15}	Kowalik	$\sum \left[a_i - \frac{x_1(b_i^2+b_i x_2)}{b_i^2+b_i x_3+x_4}\right]^2$	$[-5,5]^4$

表 5. ベンチマーク関数:bench2

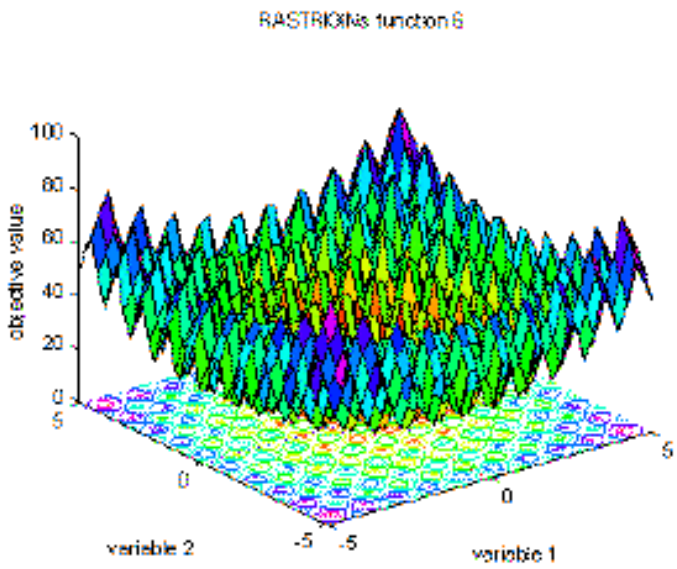
	関数名	関数式	初期値の範囲
f_1	Shifted Sphere	$\sum z_i^2+bias,\ z=x-o$	$[-100,100]^{30}$
f_2	Shifted Schwefel's Problem 1.2	$\sum_i(\sum_j z_j)^2+bias,\ z=x-o$	$[-100,100]^{30}$
f_3	Shifted Rotated High Conditioned Eliptic	$\sum(10^6)^{\frac{i-1}{n-1}} z_i^2+bias,\ z=(x-o)*M$	$[-100,100]^{30}$
f_4	Shifted Schwefel's Problem 1.2(+Noise)	$\sum_i(\sum_j z_j)^2*(1+0.4\lvert(N(0,1))\rvert)+bias,\ z=x-o$	$[-100,100]^{30}$
f_6	Shifted Rosenbrock	$\sum[100(z_{i+1}-z_i^2)^2+(z_i-1)^2]+bias,\ z=x-o+1$	$[-100,100]^{30}$
f_7	Shifted Rotated Griewank (without Bounds)	$\frac{1}{4000}\sum z_i^2-\prod\cos(\frac{z_i}{\sqrt{i}})+1+bias,\ z=(x-o)*M$	$[0,600]^{30}$
f_8	Shifted Rotated Ackley (Global Optimum on Bounds)	$-20\exp(-0.2\sqrt{\frac{1}{n}\sum x_i^2})-\exp(\frac{1}{n}\sum\cos 2\pi x_i)+20+e+bias,\ z=(x-o)*M$	$[-32,32]^{30}$
f_9	Shifted Rastrigin	$\sum[z_i^2-10\cos(2\pi z_i)+10]+bias,\ z=x-o$	$[-5,5]^{30}$
f_{10}	Shifted Rotated Rastrigin	$\sum[z_i^2-10\cos(2\pi z_i)+10]+bias,\ z=(x-o)*M$	$[-5,5]^{30}$

また、以下の図 9(a)～(f)に一部関数の形を記す.



Sphere

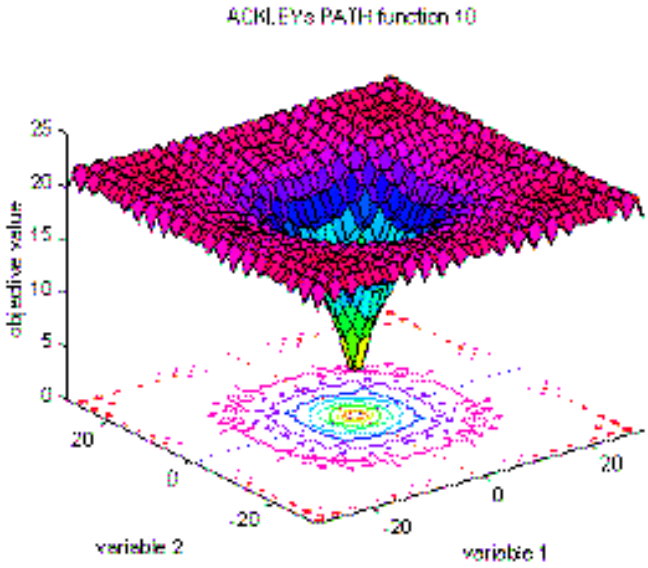
図 9(a) $\text{bench1}-f_1$



Rastrigin

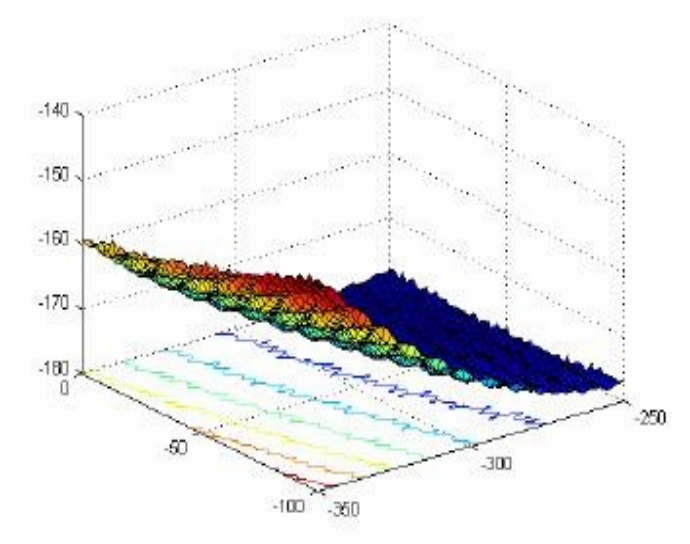
$$f(x)=\sum[x_i^2-10\cos(2\pi x_i)+10]$$

図 9(b) $\text{bench1}-f_9$



Ackley

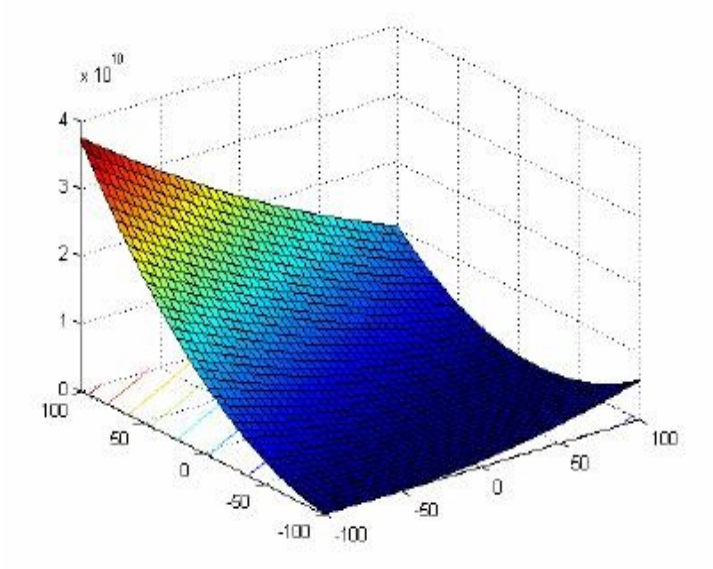
図 9(c) bench1− f_{10}



Shifted Rotated High Conditioned Eliptic

$$f(x)=\sum(10^6)^{\frac{i-1}{n-1}}z_i^2+bias,\quad z=(x-o)*M$$

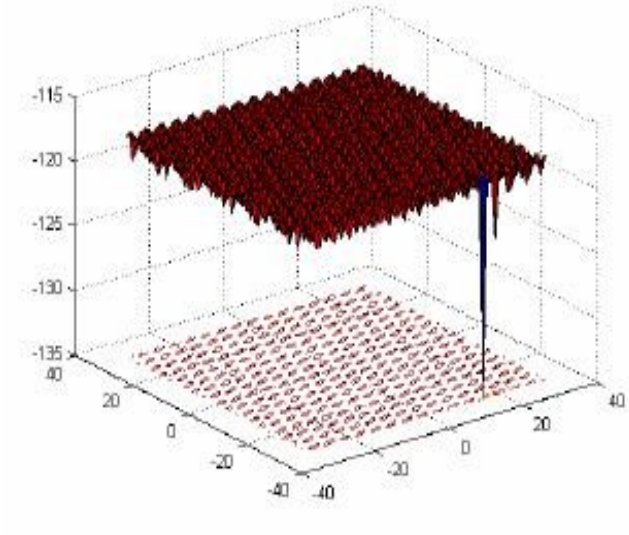
図 9(d) bench2− f_3



Shifted Rotated Griewank (without Bounds)

$$f(x)=\frac{1}{4000}\sum z_i^2-\prod\cos\left(\frac{z_i}{\sqrt{i}}\right)+1+bias\,,\ z=(x-o)*M$$

図 9(e) bench2− f_7



Shifted Rotated Ackley (Global Optimum on Bounds)

$$f(x)=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum x_i^2}\right)-\exp\left(\frac{1}{n}\sum\cos 2\pi x_i\right)+20+\mathrm{e}+bias\,,\quad z=(x-o)*M$$

図 9(f) bench2− f_8

6.2 実験の結果

それぞれの関数について、10回行った平均値の結果、および分散を表4、表5に記す.

なお、各関数はいずれも最小値を求めるものであり、結果の値が小さいものほど性能が良いことを示す. 最小値は bench1 f_8 が-1.26E+04、bench1 f_{14} が 1、bench1 f_{15} が 3.07E-04、他はすべて 0 である. bench2 関数については、いずれも bias（最小値 0 を防ぐために足しているもの）が最小値であるが、解探索過程では bias も計算に含めているものの、結果出力の段階では bias をひくことによって、最小値を 0 に補正している.

4つの手法の中で最も良い結果だったものを赤字で記した. なお、解探索においてはベンチマーク関数を紹介した表 4,5 にてそれぞれ設定されている初期値の範囲を超えないように探索を行っている。例外として、bench2 f_7 のみ大域解が初期値の範囲外にある関係で、この処理を行っていない.

表 6. bench1 の結果（左側：平均値、右側：分散）

	DE		ILSDE		旧手法		提案手法	
f_1	**3.46E+03**	2.99E+05	**3.18E+02**	1.10E+04	**2.70E+03**	3.20E+05	**1.15E+01**	1.50E+01
f_2	**2.46E+01**	8.77E+00	**8.28E+00**	1.61E+00	**2.15E+01**	4.10E+00	**1.50E+00**	3.79E-02
f_3	**3.99E+04**	3.14E+07	**3.87E+03**	7.63E+05	**2.83E+04**	3.83E+07	**3.13E+02**	2.89E+04
f_4	**6.90E+01**	2.53E+01	**5.19E+01**	1.75E+02	※	※	**8.37E+00**	3.45E+00
f_5	**1.73E+06**	1.20E+12	**3.32E+04**	4.70E+08	**5.75E+05**	5.81E+10	**6.65E+02**	1.93E+05
f_6	**3.43E+03**	6.18E+05	**3.51E+02**	7.64E+03	**2.16E+03**	3.49E+05	**1.23E+01**	1.02E+01
f_7	**1.05E+00**	9.43E-02	**1.80E-01**	4.12E-03	**6.89E-01**	3.45E-02	**6.95E-02**	3.54E-04
f_8	**-4.43E+03**	5.00E+04	**-4.71E+03**	7.03E+04	**-4.66E+03**	2.23E+05	**-6.19E+03**	7.02E+05
f_9	**2.60E+02**	1.11E+02	**2.43E+02**	3.84E+02	**2.52E+02**	2.52E+02	**2.14E+02**	1.41E+02
f_{10}	**1.29E+01**	8.01E-01	**5.78E+00**	3.39E-01	**1.13E+01**	6.43E-01	**2.50E+00**	1.65E-01
f_{11}	**3.07E+01**	1.54E+01	**4.10E+00**	7.92E-01	**2.22E+01**	4.37E+01	**1.09E+00**	4.21E-04
f_{12}	**2.07E+05**	1.84E+10	**1.80E+01**	4.83E+01	**2.12E+04**	1.28E+09	**3.49E-01**	1.79E-01
f_{13}	**2.31E+06**	2.90E+12	**3.35E+03**	2.37E+07	**8.02E+05**	1.43E+12	**1.36E+00**	4.08E-01
f_{14}	**1.00E+00**	0.00E+00	**1.00E+00**	0.00E+00	**1.20E+00**	1.60E-01	**1.20E+00**	3.59E-01
f_{15}	**4.37E-04**	7.13E-08	**5.13E-04**	1.30E-07	**7.08E-04**	1.56E-07	**8.92E-04**	5.25E-08

※旧手法では bench1 f_4 の結果が出なかった.

表 7. bench2 の結果（左側：平均値、右側：分散）

	DE		ILSDE		旧手法		提案手法	
f_1	**2.22E+03**	2.30E+05	**2.14E+02**	1.52E+03	**2.15E+03**	1.99E+05	**1.67E+01**	1.68E+01
f_2	**2.38E+04**	2.47E+07	**1.07E+04**	6.44E+06	**2.64E+04**	2.77E+07	**9.38E+03**	2.83E+06
f_3	**1.31E+08**	1.27E+15	**3.86E+07**	1.95E+14	**1.21E+08**	1.92E+15	**4.76E+07**	3.29E+14
f_4	**3.46E+04**	3.00E+07	**1.87E+04**	3.00E+07	**3.40E+04**	2.51E+07	**1.41E+04**	2.78E+07
f_6	**1.13E+08**	4.34E+15	**1.39E+06**	1.18E+12	**4.66E+07**	8.01E+14	**7.27E+04**	9.56E+08
f_7	**6.73E+02**	9.04E+03	**7.26E+01**	6.21E+02	**6.50E+02**	2.37E+04	**2.90E+01**	7.55E+01
f_8	**2.11E+01**	5.31E-03	**2.11E+01**	4.20E-03	**2.11E+01**	2.38E-03	**2.11E+01**	6.56E-03
f_9	**2.31E+02**	3.01E+02	**2.16E+02**	1.39E+02	**2.26E+02**	1.77E+02	**2.04E+02**	1.29E+02
f_{10}	**2.75E+02**	3.11E+02	**2.39E+02**	1.44E+02	**2.67E+02**	2.78E+02	**2.24E+02**	1.51E+02

※ f_2　f_3　f_4　f_8 は t 検定（両側検定、有意水準 5%）によって、ILSDE と提案手法の間に有意差がないと判断された

また、各関数の結果のグラフは次ページ以降の図 9～13 に示した.
各グラフにおいて、それぞれの手法は

- 青線　DE
- 緑線　ILSDE
- 赤線　旧手法
- 橙線　提案手法

となっている.
いずれも横軸が関数の評価回数、縦軸が評価値（10回の平均値）となっている. 縦軸については、数値でかかれているものが線形、累乗のオーダーでかかれているものが log スケールとなっている. すなわち、 bench1−f_4 、 f_8 、 f_9 、 f_{10} 、 f_{15} および bench2−f_8 、 f_9 、 f_{10} が線形スケールであり、残りは全て対数となっている.

6.3 結果の考察

表 6、7 から、ほとんどの関数で性能の良い順に、提案手法、ILSDE、旧手法、DE となるという結果が得られた. 旧手法では通常の DE より若干良い程度だったが、改善を施したことにより性能は大きく向上し、ILSDE よりも高い性能を示すようになったことが分かる.

以下、各関数について詳細に分析する.

(1) bench1−$\mathbf{f_1}$～$\mathbf{f_7}$

これらはシフトや回転のない、純粋な単峰性関数である.

結果のグラフは図 10 参照。

図10から、f_4 、f_7 をのぞき、グラフはいずれも同じ様な形となっていることが分かる. いずれも提案手法が一番良い結果を示し、以下 ILSDE、旧手法、DE と続いている. これらの関数に対して、提案手法は強力に働いたことが分かる.

f_7 についても、結果は同様である. この関数はノイズつきであるが、本手法はノイズがあっても、問題ないことが示された. しかし、グラフの傾きについては、提案手法が評価回数 6000 を超えたあたりから、緩やかなものになっている.

この原因については、まず現在の値が 0.1 と大域解 0 に十分近づいたため、ほぼ収束したことが上げられる. しかし、より理想的には、この状態からも 0.01、0.001、・・・とさらに 0 に近くなっていくことが望ましい.

提案手法は、この点については問題があると思われる. すなわち、答えに近い値をとるようになったとき、それ以降も適切に進化を続けていくようにはなっていない. その理由として、pselect の値設定が不十分であることがあげられる. 集団全体が解に近い値になり、1 世代あたり進化する個体数が少なくなっても、pselect の値は変化させていない。このため、進化しないケースでも評価を行うことになってしまっているのではないかと考えられる。

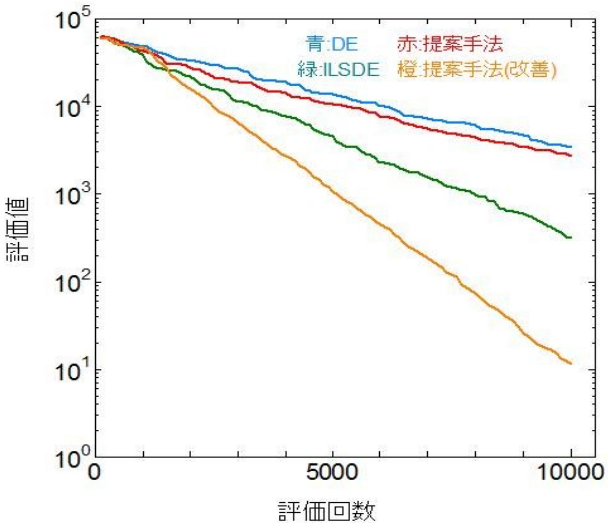
また、詳細なことは分かっていないが、提案手法において、ある程度世代が進むと個体が偏ってしまうケースがあることを確認している. 分類精度が良くないことが原因ではないかと考えられる. DE は個体にある程度のランダム性があるからこそ、局所解に陥らず解探索を行える。個体が偏ってしまうと、適切に進化しなくなってしまう恐れがある. この個体選択の偏りも、終盤の解探索に悪影響を及ぼし、収束速度の低下を招いてる可能性がある.

f_4 については、旧手法では答えが出なかった. 分類時に負例判定ばかり出るようになり、進化が止まってしまったためである. 今回の実験結果では、答えが出なかったのは f_4 だけだが、他にもいくつかの関数でこの現象が発生することを確認している. これは学習例の偏りと分類手法が単純な kNN であることが原因だと思われる。改善した提案手法では、学習例の偏りを考慮に入れた分類手法をとり、必ず一定数正例判定するよう設定したため、このような問題は起こらなくなった。

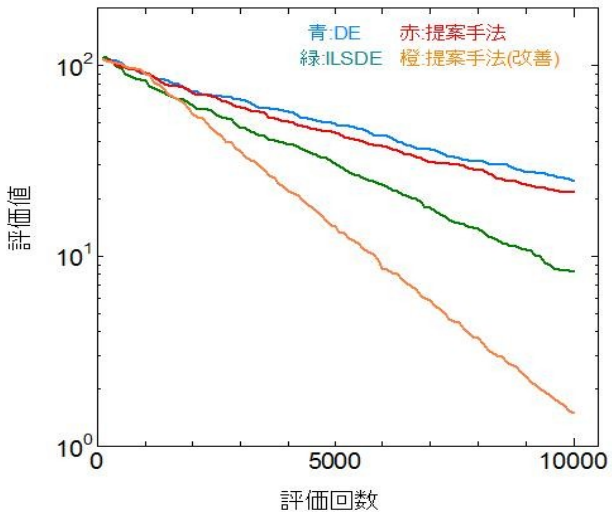
また、この関数に対して提案手法は他よりも特によい結果を残している. この関数は表にあるとおり、ベクトルの各次元の値の中で最も絶対値が大きい値をその答えとするという単純なものである. すなわち、すべての次元で同様に低い値となれば、より良い結果が出る.

この関数に対して強いということは、本手法は解のすべての次元の値を一緒となるようなケース

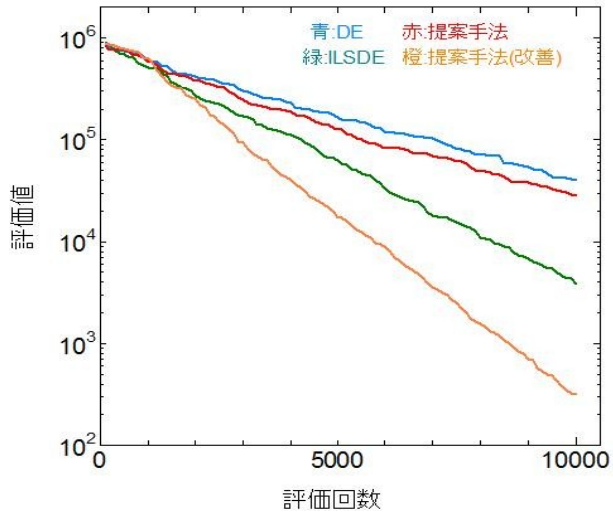
で、特に優れていると考えられる. 理由として、機械学習がこのような、解が分かりやすい場合、とても効果的に働いているからだと推測できる。bench1 では、 f_4 に限らず、 f_{15} 以外全てで大域解における全次元の値が一緒である. そのため bench1 では、本手法の性質から、特に良い結果が出ていると考えられる。



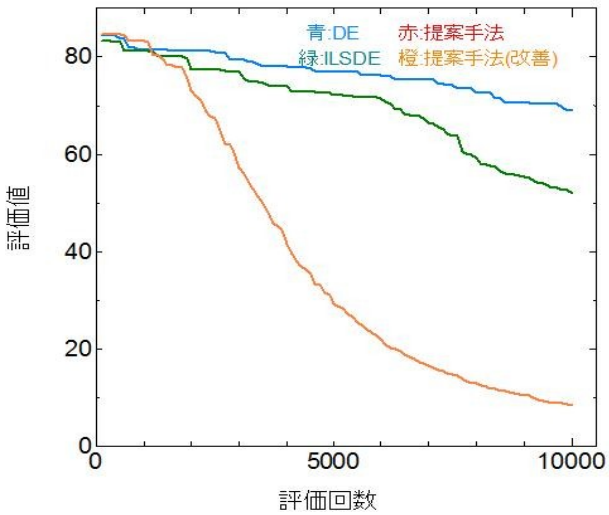
(a) bench1−$\mathbf{f}_1$



(b) bench1−$\mathbf{f}_2$

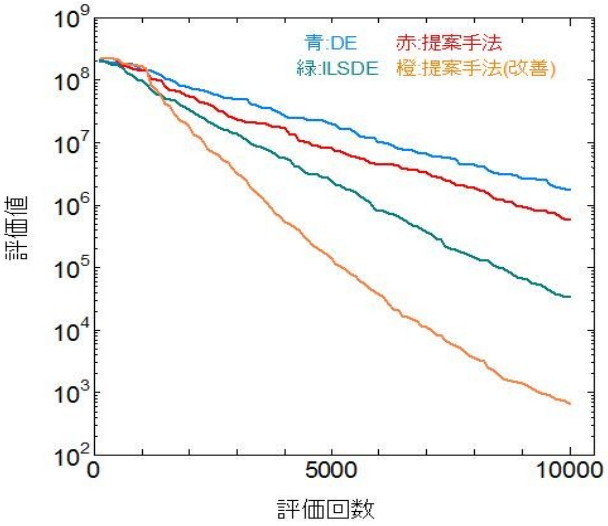


(c) bench1−$\mathbf{f}_3$

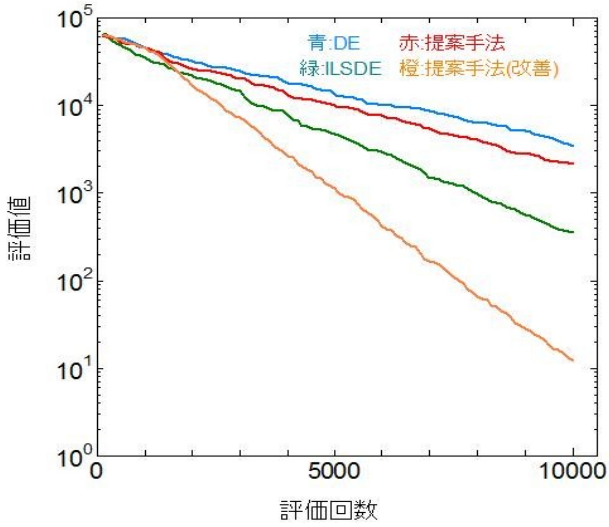


(d) bench1−$\mathbf{f}_4$

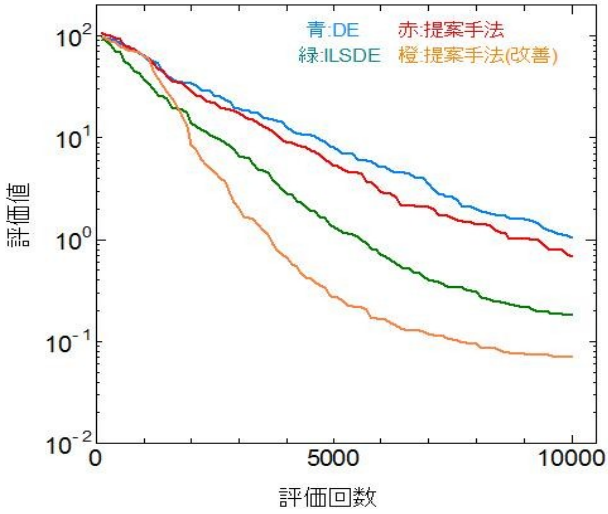
図 10. 実験結果グラフ bench1−$\mathbf{f}_1$～$\mathbf{f}_7$ ($\mathbf{f}_1$～$\mathbf{f}_4$ まで)



(e) bench1−$\mathbf{f}_5$



(f) bench1−$\mathbf{f}_6$



(g) bench1−$\mathbf{f}_7$

図 10. 実験結果グラフ bench1−$\mathbf{f}_1$～$\mathbf{f}_7$ ($\mathbf{f}_5$～$\mathbf{f}_7$ まで)

(2) bench1−f_8～f_{13}

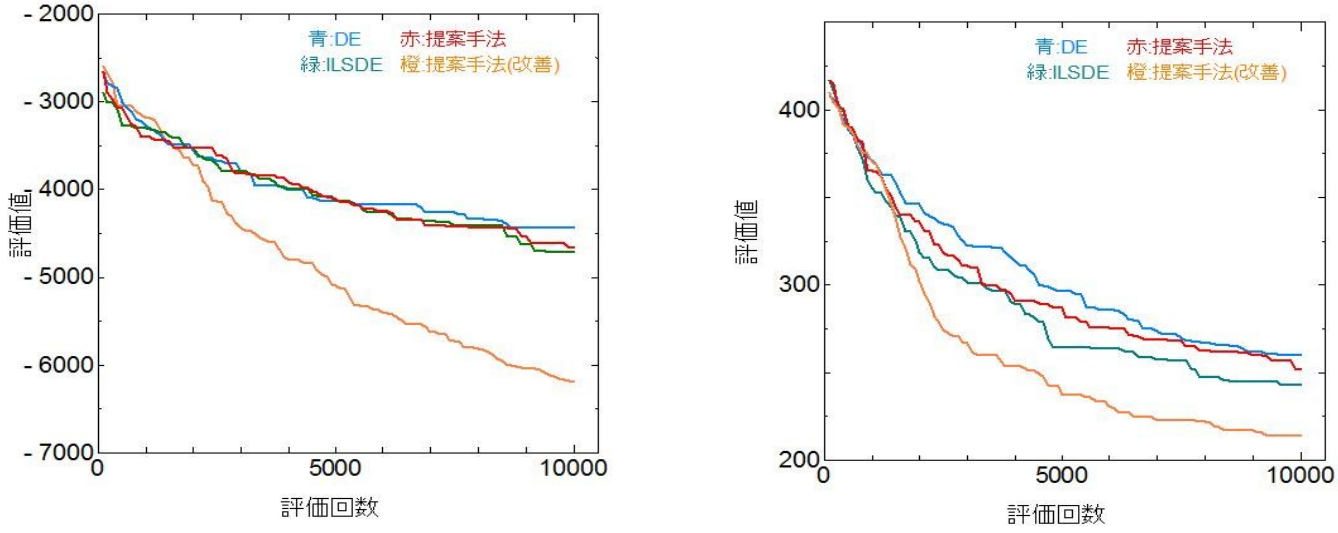
これらはシフトや回転のない、純粋な多峰性関数である. グラフは図 11(a)～(f).

いずれも提案手法が1番良い性能を示している. 特に f_8 では、ILSDE が DE よりわずかに良くなっているのに対し、提案手法では DE より性能がかなり高くなっている. 多峰性関数についても、本手法が ILSDE より良い性能を示していることが確認された.

f_{12} 、 f_{13} について、図(l)、(m)から、評価回数が 5000 前後で値が 100 より小さくなったところから、グラフの傾きが小さくなり、以降 DE と傾きがほぼ平行な状態で推移していることが分かる. これは、収束速度が DE と変わらなくなったことを示す. これは(1)の f_7 の時と同様、p_select の値設定が適切でなく、負例とすべきものの多くを正例判定してしまっていること。また集団の個体が偏ってしまっていることが原因ではないかと推測される。大域解は 0 であるため、まだ値が 100 前後の時点でこのような現象が起きてしまうのは問題があると言える. 進化が一定度進むと、性能低下が見られるのが本手法の欠点であることが、再び示された.

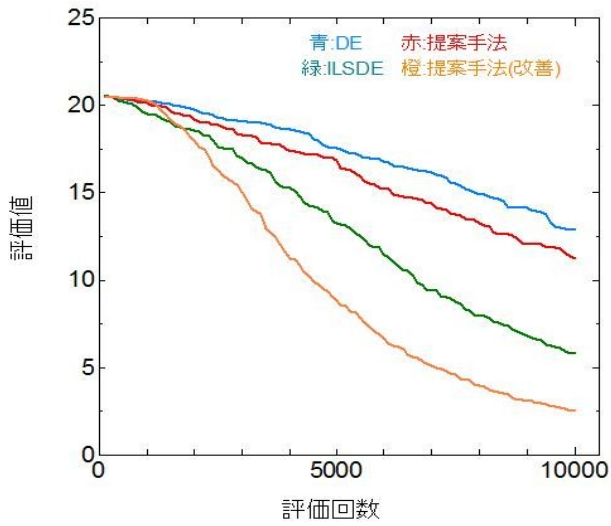
f_9 について。この関数は Rastrigin 関数であり、図からわかるように非常に凹凸が多い関数である. DE のような探索手法では局所解に陥りやすくなっており、DE が苦手とする関数と言える。

このような関数においても、提案手法は最も良い性能を示していることが分かる. しかし、途中からは解探索速度は落ちてしまっている。通常の DE の場合、途中から1世代あたり 100 個体中わずか数個しか進化しなくなるような関数であるため、本手法においても厳しい結果が出るのは仕方がない面もある. この関数に対しても一定以上の成果を上げることが、本研究の大きなテーマのひとつとなっている。

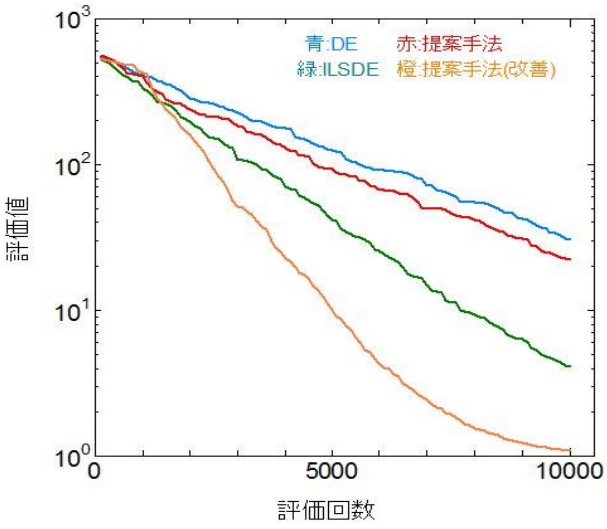


(a) bench1−f_8　　(b) bench1−f_9

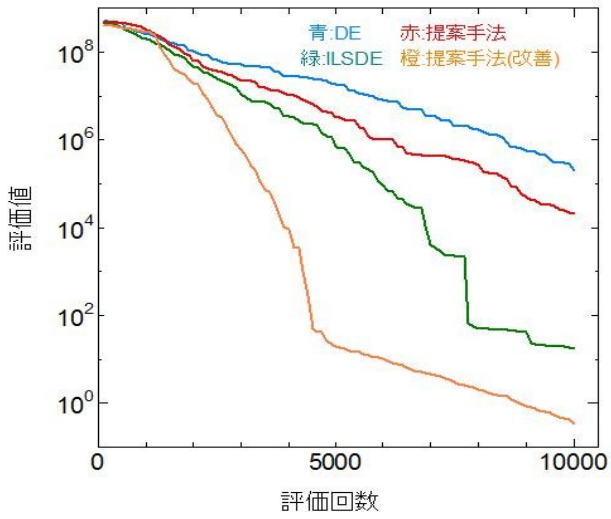
図 11. 実験結果グラフ bench1−f_8～f_{13} (f_8, f_9)



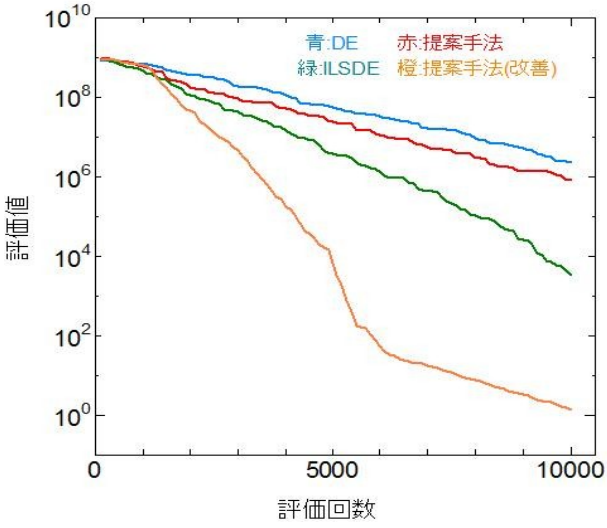
(c) $\mathrm{bench1}-\mathbf{f}_{10}$



(d) $\mathrm{bench1}-\mathbf{f}_{11}$



(e) $\mathrm{bench1}-\mathbf{f}_{12}$



(f) $\mathrm{bench1}-\mathbf{f}_{13}$

図 11. 実験結果グラフ　$\mathrm{bench1}-\mathbf{f}_{8}\sim\mathbf{f}_{13}$　（　$\mathbf{f}_{10}\sim\mathbf{f}_{13}$　）

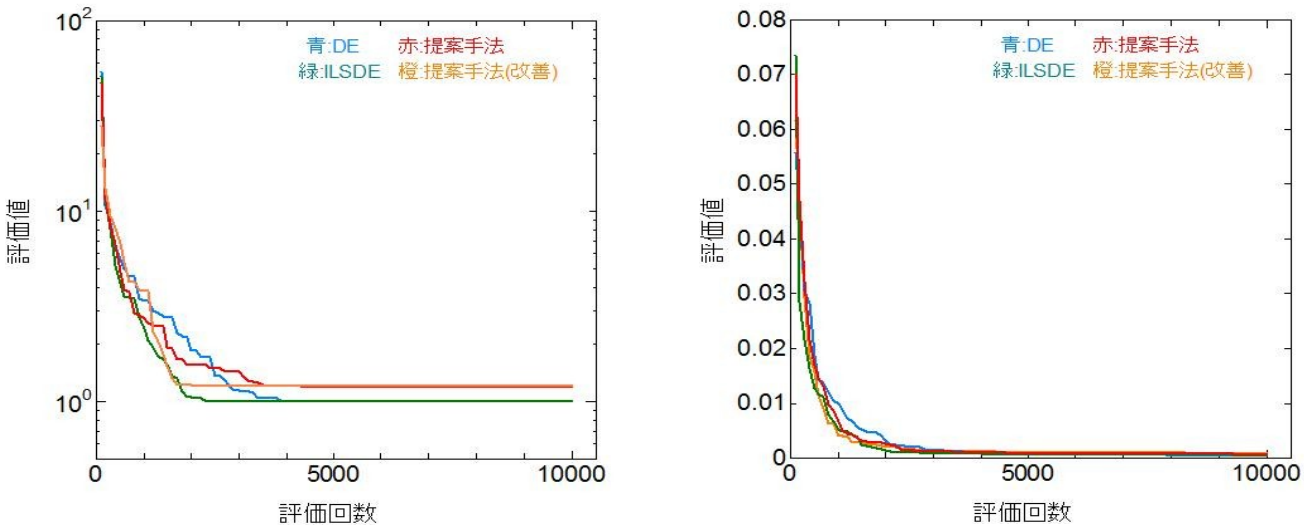
(3) bench1－$\mathbf{f_{14}}$～$\mathbf{f_{15}}$

これらの関数も(2)同様、シフトや回転のない多峰性関数であるが、(2)と違い、ベクトルの次元数が少ない．(2)では次元数30であるのに対し、 f_{14} は次元数2、 f_{15} は次元数4である．

実験結果グラフは図 12。

次元数が少ないこともあり大域解に至るのが容易で、すべての手法で 2000～3000 回程度でほぼ大域解に到達している．

収束速度の点でも、多少 DE が他より劣っている程度で、大きな差は見られない．しかし、この関数において、提案手法は大域解の前で収束してしまう現象が見られた．そのため、この2つの関数に関しては、提案手法の結果は DE より結果が悪くなっている．ほとんど解に近い部分での話ではあるが、このような大域解に到達前に収束してしまうのは、(1) f_7 で述べたとおりであり、個体に偏りが生じてしまっているためであると考えられる．提案手法には、解に大きく近づいた際の挙動に問題があることが本結果からも示されている。



(a) bench1－$\mathbf{f_{14}}$　　(b) bench1－$\mathbf{f_{15}}$

図 12. 実験結果グラフ bench1－$\mathbf{f_{14}}$, $\mathbf{f_{15}}$

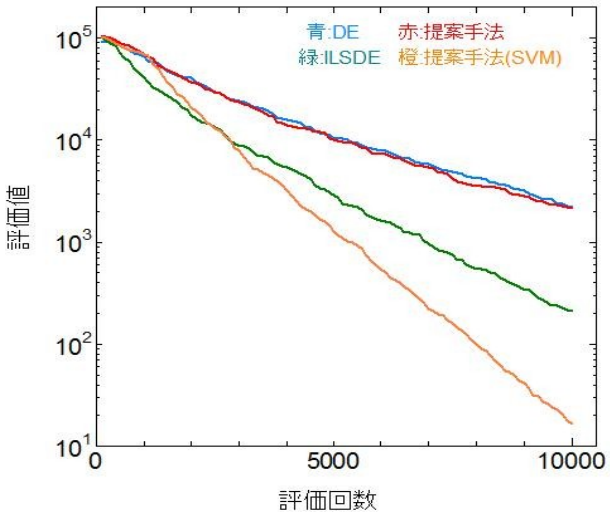
(4) bench2-$\mathbf{f_1}$～$\mathbf{f_4}$

これらはシフトや回転のある、単峰性の関数である.
実験結果のグラフは図 12(a)～(d)。

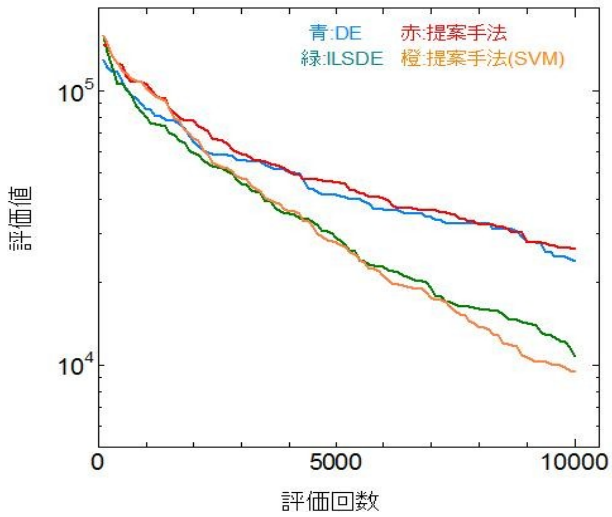
図から、f_1 については提案手法がもっとも高い結果を得たものの、f_2 ～ f_4 の間ではほとんど変わらない結果となった. f_2 、f_4 では平均値が ILSDE をやや上回っているが、f_3 は少し劣っており、有意水準 5%の t 検定では両者に有意差はないという結果になった。bench1 の類似関数と比べても明らかに結果は落ちており、今回の提案手法が、シフト・回転した関数に対しては弱いことが見てとれる.

性能が落ちる原因としては、(1) f_4 の所で考察したように、本手法は全ての次元を近い値にする点に優れているためであると考えられる. 今回の関数では、大域解の位置をシフトしているため、大域解となるベクトルの値は全ての次元で近い値にはならない. よって、(1)の時は、大域解となるような次元が一緒であるために良くなっていた部分がなくなり、性能が落ちてしまったと考えられる. 現実問題としては、(1)のように全部の次元で同じ値の時に大域解となるよりは、今回のような各次元バラバラの値が大域解となるケースが多い. 実用上では全次元が同じような値の時に強いという特性のメリットは受けづらいと考えられるため、今後改善していく必要がある部分だといえる、

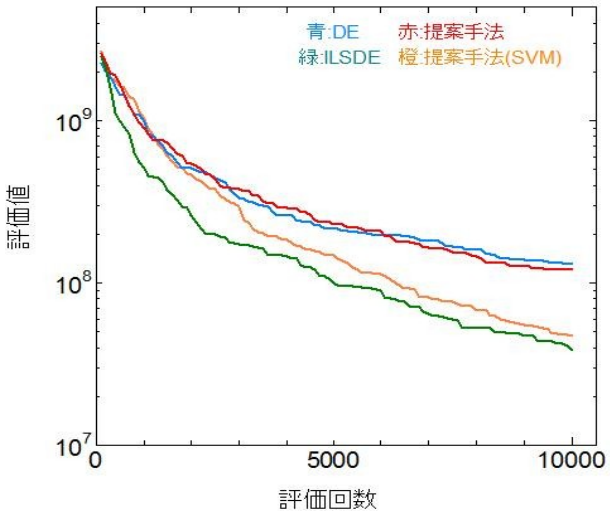
しかし同時に解がシフトしたようなケースでも、LSDE と同等以上の性能は確保出来ていると言える. 旧手法では、DE と同程度まで性能が落ち込んでしまっているが、改善を施したことによりシフト関数においても ILSDE 以上にの性能まで向上させることが出来たといえる。



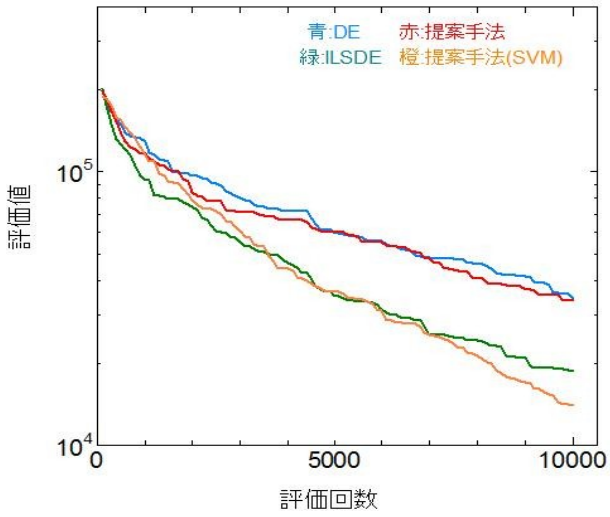
(a) bench2−$\mathbf{f}_1$



(b) bench2−$\mathbf{f}_2$



(c) bench2−$\mathbf{f}_3$



(d) bench2−$\mathbf{f}_4$

図 12. 実験結果グラフ　bench2−$\mathbf{f}_1$～$\mathbf{f}_4$

(5) bench2−$\mathbf{f_6}$～$\mathbf{f_{10}}$

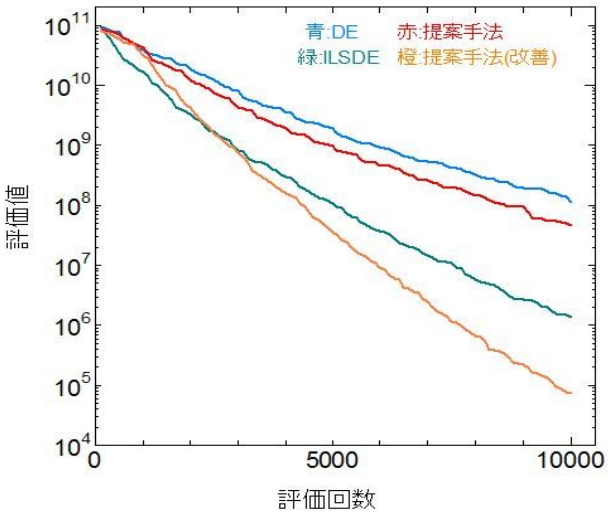
これらはシフトや回転のある、多峰性の関数である.

実験結果のグラフは図 13(a)～(e)。

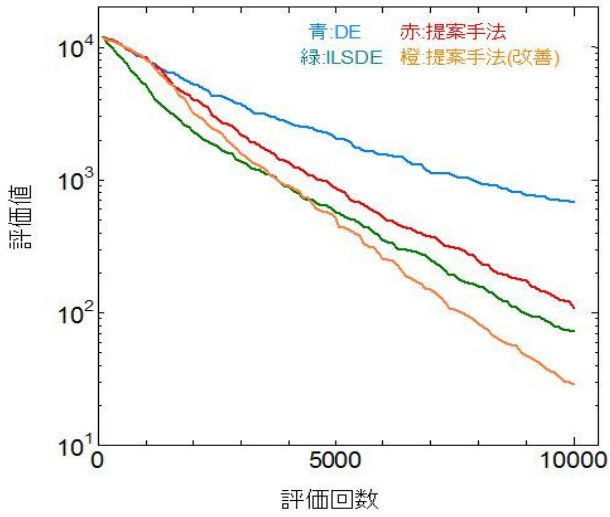
f_6 、 f_7 では、提案手法が最も良い結果を得ている. 多峰性関数についても、難解なものでなければ、シフトや回転による性能落ちはあると考えられるが、収束速度は2つとも ILSDE より大きく、提案手法は ILSDE よりも高い性能を示している.

f_8 、 f_9 、 f_{10} は局所解が多く、DE で解くのが難しい関数である. 特に f_8 はいずれの手法も DE とほとんど変わらない結果となっていしまっている. この関数の最小値は(bias をのぞいているため)0 であるため、値が 21 付近で収束しているのは、大域解ではなく局所解にはまってしまっている状態である. この関数 f_8 の関数自体の図を見ても、解にたどりつくのが非常に困難であることが伺える。

f_9 、 f_{10} は Rastrigin 関数のシフト、およびシフト＋回転を加えた関数であるため、これも DE にとっては難易度の高い関数である. これについても、提案手法が一番いい値を出してはいるものの、 bench1−f_9 の時と同様、最小値が 0 であるのに値が 200 付近の時点で局所解に陥いつつある様子が見てとれる. f_8 も含め、DE が苦手とする関数については、本手法でも苦手のままであることが、改めて示された. DE の探索手法でこれらの関数で大域解にたどり着くのは、相当な性能改善が必要であり、提案手法ではまだまだ性能不足であると言える。

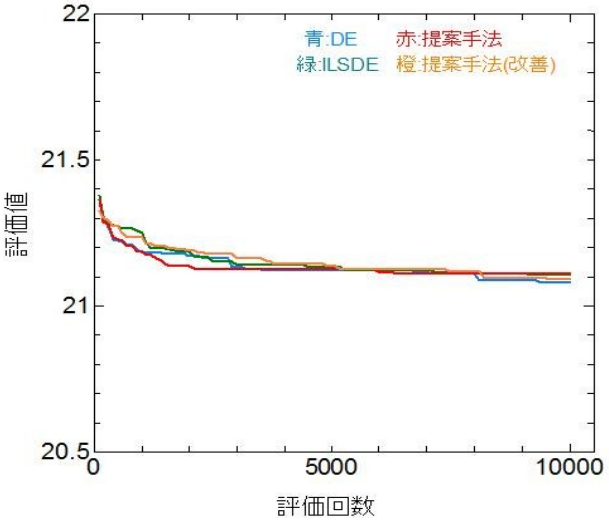


(a) bench2−$\mathbf{f_6}$

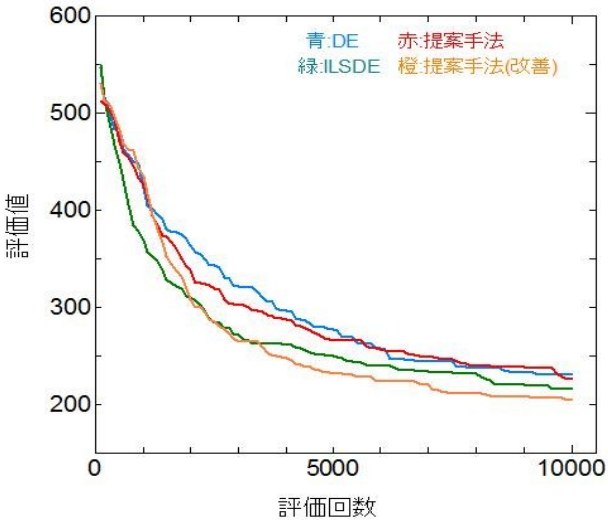


(b) bench2−$\mathbf{f_7}$

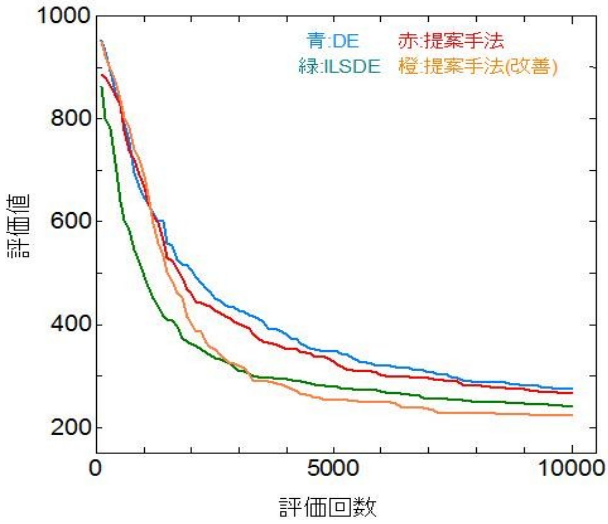
図 13. 実験結果グラフ bench2−$\mathbf{f_6}$～$\mathbf{f_{10}}$ ($\mathbf{f_6}$, $\mathbf{f_7}$)



(c) bench2−$\mathbf{f}_8$



(d) bench2−$\mathbf{f}_9$



(e) bench2−$\mathbf{f}_{10}$

図 13. 実験結果グラフ　bench2−$\mathbf{f}_6$～$\mathbf{f}_{10}$（ $\mathbf{f}_8$～$\boldsymbol{f}_{10}$ ）

6.4 SVC-DE との性能比較

前節の結果から、提案手法は一部問題点は見られたものの、全体的には先行研究の一つである ILSDE よりも高い性能を得ることが出来た。

次に、より新しい先行研究である SVC-DE と提案手法の性能比較を行う。ベンチマーク関数は bench2 の f_5 、 f_7 を除いた f_1～f_{10} の 8 個の関数で行う。

SVC-DE のプロットデータを所持していないため、グラフによる比較は行わず、関数評価 1 万回終了時のデータとの比較のみになる。結果は表 8。

表 8. SVC-DE との比較（左側：平均値、右側：分散）

	DE		提案手法		SVC-DE	
f_1	**2.22E+03**	2.30E+05	**1.67E+01**	1.68E+01	**1.07E-01**	1.35E-03
f_2	**2.38E+04**	2.47E+07	**9.38E+03**	2.83E+06	**3.54E+03**	1.77E+06
f_3	**1.31E+08**	1.27E+15	**4.76E+07**	3.29E+14	**1.80E+07**	3.31E+13
f_4	**3.46E+04**	3.00E+07	**1.41E+04**	2.78E+07	**7.71E+03**	7.67E+06
f_6	**1.13E+08**	4.34E+15	**7.27E+04**	9.56E+08	**2.54E+03**	9.67E+06
f_8	**2.11E+01**	5.31E-03	**2.11E+01**	6.56E-03	**2.08E+01**	4.37E-03
f_9	**2.31E+02**	3.01E+02	**2.04E+02**	1.29E+02	**2.09E+02**	1.71E+02
f_{10}	**2.75E+02**	3.11E+02	**2.24E+02**	1.51E+02	**2.15E+02**	1.88E+02

ILSDE との性能比較ではほぼ性能が上だった提案手法だが、より新しい先行研究である SVC-DE との比較では一歩劣るという結果であった。

特に f_1 、 f_6 については SVC-DE の方が評価値が 1 桁以上良いという結果が出た。最も単純な f_1 と多峰性関数の中では簡単である関数 f_6 において特に良い結果が出ているため、SVC-DE による評価値予測は、関数が簡単な時は特に大きな効果を上げると推察出来る。

その他について。 f_2～f_4 については、上記 2 つの関数ほど大きな差はついていないが、提案手法よりも SVC-DE の方が 2～3 倍良い評価値となっている。 f_8～f_{10} については、複雑な関数であることもあってか、SVC-DE と提案手法の間にはほぼ有意差はない状態となった。

このことから、次のことが言える。現時点で、提案手法が SVC-DE より劣っているのは間違いない。しかし、関数の難易度が上がるにつれ、その差は縮まってきている。簡単な関数において、SVC-DE は高い効果を上げているが、より複雑な関数では効果は落ちてきてる。

本提案手法は一定の性能改善に成功したが、より良い個体選択の作成という点ではまだまだ改善の余地が大きい。今後の改善結果次第では、本手法が SVC-DE と同等かそれ以上の性能となる可能性も十分あると考えている。

7 まとめ

本論文では、最適化問題に使用されるアルゴリズム、DE(差分進化)について、その性能改善のため機械学習を導入した手法を提案した。

DE は扱いが簡単かつ高い精度・高速性を持つアルゴリズムであるが、その性質から関数の評価回数が多くなるという問題がある。このため、実用上で関数の評価コストが高い場合、評価回数の多さから DE 全体の性能が落ちてしまう。したがって、この多い DE の関数評価回数を削減することが DE の性能改善に繋がることを述べた。

このような関数の評価回数削減のための先行研究、ILSDE と SVC-DE を紹介した。そして DE の Mutation 時に機械学習による分類を導入するという提案手法、およびそのさらなる改善のために取り組んだ各種手法について説明した。その後、提案手法で複数のベンチマーク関数による性能比較実験にて、SVC-DE にはやや劣るものの、提案手法が通常の DE からは大きく改善され、ILSDE 以上の性能があることを示した。

8 今後の課題

本提案手法はDEの性能改善において、一定の成果を示した。しかしながら、先行研究であるSVC-DEに劣っており、さらなる性能向上のためには、まだまだ改善点も多い。本章ではこの点を踏まえ、今後の課題について述べる。

(1)分類精度の改善

本論文では分類精度向上の重要性を述べ、そのために学習例の収集方法、分類手法、特徴ベクトルのとり方について様々な工夫した点を述べた。しかしながら、現状ではまだ分類の精度は高いとは言えない状態である。

現時点での提案手法の分類精度、すなわち正例と判定されたものが本当に正例と判定される割合について調査した所、 $\mathrm{bench2}\text{-}f_1$ のような比較的精度が高かったケースでも30%前後であったことを確認している。 精度が低い場合（Rastrigin関数の終盤など）では、精度が10%を切っているような場面も見られた。分類精度の改善は、今後の課題の中でも特に優先度の高いものであるといえる。

改善するための候補として、学習例の収集方法をさらに工夫することが考えられる。

データ自体が進化していくために、学習例にも動的更新が必要なことはすでに述べた通りであるが、現在の手法で適用している、新しいサンプルを学習例群に逐次蓄積していくだけでは不十分ではないかと考えている。学習例の正例・負例は常に現在の分類基準にふさわしいものである必要があり、 進化していくデータに即した学習例を用意しなければならない。そのため、現在の世代の分類基準を適切に判断し、場合によっては既存の学習例の正例・負例を書き換えるような、さらなる動的な学習例収集方法が求められている。

また、学習例の正例・負例の割合が偏ってしまってる点についても、本研究に置いては良い補正方法が見つからずそのままの状態で使用したが、適切に補正をかけた方が、より有用な学習例になると考えられる。上記の進化に即した学習例を用意するのとあわせ、ただ単に学習例を集めるだけでは、常に分類基準が変わる本研究の分類用の学習例としては不十分であり、偏りや現在の進化状況から適切な補正をかける必要がある。

学習例の収集方法の改善は、本研究のさらなる性能向上のためには、非常に重要な課題であると考えられる。

その他の部分、つまり分類手法・特徴ベクトルのとり方についても、現在の方法が最善であるとは考えていない。どちらも比較的単純な方法で行っており、より複雑な分類手法・特徴ベクトルを使えば、さらなる性能改善に繋がる可能性は大いにある。

しかしながら、ただ高度な手法を使うだけでは効果は出ないと思われる。分類手法ではSVMが、特徴ベクトル作成では親ベクトル情報を付加することが失敗に終わったように、本研究における、分類対象の正例・負例の基準が常に変化し続けるという特殊性から、通常の分類器学習と同じアプローチでは上手くいかない部分もある。これらの方法についても、学習例の場合と同様、分類対象が進化していくことを念頭に置いて、良い方法を試行錯誤していくことが要求されている。

(2)良い個体選択例の作成

現在の提案手法においては、Mutation 段階で進化しないと予想したもの、すなわち悪い個体選択を切り捨てているのみである。やっていることは、分類器の導入場所が Selection と Mutation という違いはあるものの、SVC-DE とかなり近くなってしまっている状態である。

本手法にて Mutation で分類を行ったそもそもの理由は、今回のように不要な関数評価を減らすというのももちろんあるが、4.1 節 ILSDE と同様、Mutation 時により良い個体選択を作成したいという目的があった.

残念ながら今回はただ分類するだけに終わってしまったが、より分類精度が上がってくれば、個体選択の強化も可能になっていくると考えている. 例えば Mutation における個体選択を、ただランダムに行うのではなく、非常に良い個体と分類されるようなものを作成する。そのような個体が常にベクトル集団に一定数存在するよう、個体選択を操作することが出来れば、DE の収束速度は大きく上がる可能性がある。また場合によってはあえて悪い個体選択例も生成することで、より局所解に陥ることをなくすことも可能である。

現在の分類精度では中々難しい課題ではあるが、分類精度の向上次第では、より優れた個体選択例を作れるようになり、Rastrigin 関数のような、既存の DE の探索法では厳しい物であっても、上手く解決出来るようになる可能性がある。現状では SVC-DE とあまり大差ない手法に留まってしまっているが、さらに発展させていけば先行研究と十分に差がつけられる手法であると考えている。

(3)実用上の応用

本研究の前提として、関数の評価回数削減が必要になるほど、関数の 1 回あたりの評価コストが非常に大きいというものがある。1 回あたりの評価コストがとても小さいもベンチマーク関数でいくら評価回数を削減しても、評価コストが大きいケースに応用出来なければ意味がない。本研究のゴールは、何らかの実用上の問題に本提案手法を適用することである。

現在の提案手法の性能では、まだ実際に応用するにはいたらない段階であり、まずは(1)(2)の課題を解決していくことが先決だと思われる。

しかし、最終的には実用上の問題、例えば検索エンジンのランキング学習[14]において本手法を使用し結果を出すことが、本研究の大きな最終課題として存在していると考えたため、今後の課題として記した。

参考文献

[1] Rainer Storn and Kenneth Price. Differential evolution- a simple and efficient heuristic global optimization over continuous spaces. *Journal of Global Optimization,* 11, pp.341-459, 1997.

[2] Yuan Shi, Zhen-zhong Lan and Xiang-hu Feng Differential Evolution based on improved learning strategy, *PRICAI 2008,* pp.880-889, 2008.

[3] Lu, Xiaofen, Ke Tang and Xin Yao Classificationassisted differential evolution for computationally expensive problems. *Evolutionary Computation(CEC),2011*, 2011.

[4] Xin Yao, Yong Liu and Guangming Lin. Evolutionary Programming Made Faster. *IEEE Trans. on Evolutionary Computation, vol. 3*, pp. 82-102 ,1999.

[5] P.Suganthan, N.Hansen, J.Liang, K.Deb, Y.Chen, A.Auger and S.Tiwari Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization *KanGAL Report 2005005 (2005)*, 2005.

[6] Price, Kenneth, Rainer M. Storn, and Jouni A. Lampinen. Differential evolution: a practical approach to global optimization. *Springer, 2006*, 2006.

[7] Jin, Yaochu. A comprehensive survey of fitness approximation in evolutionary computation. *Soft computing 9.1,*pp.3-12,2005.

[8] Farina, Marco, and Jan K. Sykulski. Comparative study of evolution strategies combined with approximation techniques for practical electromagnetic optimization problems. *Magnetics, IEEE Transactions on 37.5,* pp.3216-3220, 2001.

[9] Hajela, P., and J. Lee. Genetic algorithms in multidisciplinary rotor blade design. *Proceedings of the 36th AIAA Structures, Structural Dynamics, and Materials Conference,* 1995.

[10] Holland, John H. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. *U Michigan Press*, 1975.

[11] Kennedy, James, and Russell Eberhart. Particle swarm optimization. *Proceedings of IEEE international conference on neural networks. Vol. 4. No. 2*, 1995.

[12] Karaboga, Dervis. An idea based on honey bee swarm for numerical optimization. Vol. 200. *Technical reporter06, Erciyes university, engineering faculty, computer engineering department*, 2005.

[13] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning, vol. 20, no. 3,* pp. 273–297, 1995.

[14] B.Danushka, N.Noman and Iba.H. Rankde: learning a ranking function for information retrieval using differential evolution. *Proceedings of the 13 th annual conference on Genetic and evolutionary computation*. , ACM, 2011

[15] V. Vapnik. Statistical Learning Theory. Wiley, Chichester, GB, 1998

[16]「GEATbx: Example Functions」
http://www.geatbx.com/docu/fcnindex-01.html

[17]「SVM-Light Support Vector Machine」
http://svmlight.joachims.org/