2002 年 03 月 18 日

多目的遺伝的アルゴリズム (ver 1.2) 仕様書

同志社大学工学部　知的システムデザイン研究室

廣安知之， 三木 光範， 渡邉 真也， 岡田 靖男， 奥田 環

目次

はじめに

本ドキュメントは同志社大学工学部知識工学科知的システムデザイン研究室[1]が開発した多目的遺伝的アルゴリズムプログラムの使用方法と仕様についてまとめたものである．このプログラムは，知的システムデザイン研究室の多目的研究グループのサイト[2]からダウンロードすることが可能である．

また，このプログラムは多目的 GA の手法として以下の 4 手法に対応しており，それぞれの手法に応じたコンパイルを行うことで 4 つの手法を実行することができる．

2. . **MOGA[2]**
2. . **NSGA-II [3]**
2. . **SPEA2[4,5]**
2. . **NCGA[6]**

本ドキュメントは大きく以下のような構成になっている．まず，2 章でプログラムの実行方法とテスト関数における性能について述べる．つづいて 3 章においてプログラムの詳しい解説を行い，4 章において本プログラムを用いた 2 次利用の方法について説明する．

注意：

本プログラムは，Linux 環境において g++(version 2.95.4 20010319)および mpiCC (MPICH 1.2.2, Lam 6.5.2)を用いた場合での動作確認しか行っていない．そのため，Windows 環境化におけるコンパイル・実行，version 2.95.4 以前の g++ を用いたコンパイル・実行は不明である．

[1] http://is.doshisha.ac.jp/
[2] http://mikilab.doshisha.ac.jp/dia/research/moga/

２．プログラムの使用方法

提供するファイルを用いることにより，4 種類の多目的 GA 手法とそれぞれの手法に対応する 4 種類の並列プログラム，計 8 つのプログラムを生成することができる．利用可能な多目的 GA 手法を以下に示す．

- **MOGA[2]**
- **並列 MOGA（マスタースレーブ型並列 GA)**
- **SPEA2[3]**
- **並列 SPEA2（マスタースレーブ型並列 GA)**
- **NSGA-II [4,5]**
- **並列 NSGA-II （マスタースレーブ型並列 GA)**
- **NCGA[6]**
- **並列 NCGA2（マスタースレーブ型並列 GA)**

全てのプログラムは，C++言語に基づき書かれており，一部のファイルでは **STL**(標準テンプレートライブラリ)が利用されている．また，並列プログラミングとしては MPI が用いられている．

2.1 使用ファイル

ダウンロードしたファイルを展開すると，以下のファイルが生成される．

[makefile ファイル]

makefile

[cpp ファイル]

cross_over.cpp
func.cpp
func_numerical.cpp
gene.cpp
loadconf.cpp
main_m_s.cpp
main_ncga.cpp
main_ncga_mpi.cpp
main_nsga2.cpp
main_nsga2_mpi.cpp
main_oneisland.cpp
main_spea2.cpp

main_spea2_mpi.cpp
mtrand.cpp
mutation.cpp
ncga.cpp
ncga_mpi.cpp
one_mpi.cpp
one_nsga2.cpp
one_nsga2_mpi.cpp
one_spea2.cpp
one_spea2_mpi.cpp
oneisland.cpp
pop.cpp
select.cpp
select_nsga2.cpp
select_spea2.cpp

[h ファイル]
base.h
cross_over.h
evaluat.hn.h
func.h
func_numerical.h
gene.h
loadconf.h
mtrand.h
mutation.h
ncga.h
ncga_mpi.h
one_mpi.h
one_nsga2.h
one_nsga2_mpi.h
one_spea2.h
one_spea2_mpi.h
oneisland.h
pop.h
select.h

select_nsga2.h
select_spea2.h

ファイルを展開するためには以下のコマンドをタイプする．

```
Tar xvzf ga2k.tar.gz
```

2.2 コンパイルとプログラムの実行

前述の通り，本プログラムでは MOGA，SPEA2，NSGA-II ，NCGA の 4 つの手法を実行することができる．各手法のコンパイルは，全て makefile によって行うことができる．以下，コンパイルおよびデフォルトでのコンパイルに用いる最適化オプションについて説明する．

2.2.1 **コンパイルオプション**

ここでは，make を利用したコンパイルの方法を述べる．Make を行う前に makefile を環境に応じて変更する．変更の必要があると考えられるのはコンパイラと最適化オプションである．コンパイラは標準で CC = g++ としている．また，最適化オプションは CFLAGS = -O2 –funroll-loops とした．尚，MPI コンパイラとして MPICC = mpiCC をデフォルトとして設定している．

2.2.2 **各手法別の** make **実行について**

適宜，環境に応じて makefile を変更したら下記のように各手法ごとに「make」を実行する．

- MOGA
  以下のコマンドを実行することにより実行プログラム [moga] を作成することができる．

  make

  make に成功すると moga という実行ファイルが生成される．これを実行すると標準の設定で MOGA が実行される．

  ./moga

  makefile を見てもらえれば分かるように，MOGA は，以下のファイルをコンパイルすることにより生成することができる．

  oneisland.cpp
  pop.cpp
  gene.cpp
  func.cpp

func_numerical.cpp
mutation.cpp
cross_over.cpp
select.cpp
mtrand.cpp
loadconf.cpp
main_oneisland.cpp

- MPI_MOGA
  以下のコマンドを実行することにより実行プログラム [mpi_moga] を作成することができる.

  make mpi_moga

  make に成功すると mpi_moga という実行ファイルが生成される.これを実行すると標準の設定で MOGA が実行される.

  mpirun –np 2 mpi_moga

  makefile を見てもらえれば分かるように，mpi_moga は，以下のファイルをコンパイルすることにより生成することができる.

  oneisland.cpp
  pop.cpp
  gene.cpp
  func.cpp
  func_numerical.cpp
  mutation.cpp
  cross_over.cpp
  select.cpp
  mtrand.cpp
  loadconf.cpp
  one_mpi.cpp
  main_m_s.cpp

- SPEA2
  以下のコマンドを実行することにより実行プログラム [spea2] を作成することができる.

  make spea2

  make に成功すると spea2 という実行ファイルが生成される．これを実行すると標準の設定で SPEA2 が実行される.

./spea2

makefile を見てもらえれば分かるように，SPEA2 は，以下のファイルをコンパイルすることにより生成することができる．

oneisland.cpp
pop.cpp
gene.cpp
func.cpp
func_numerical.cpp
mutation.cpp
cross_over.cpp
select.cpp
mtrand.cpp
loadconf.cpp
main_oneisland.cpp
select_spea2.cpp
one_spea2.cpp
main_spea2.cpp

- MPI_SPEA2

以下のコマンドを実行することにより実行プログラム [mpi_spea2] を作成することができる．

make mpi_spea2

make に成功すると mpi_spea2 という実行ファイルが生成される．これを実行すると標準の設定で SPEA2 が実行される．

mpirun –np 2 mpi_spea2

makefile を見てもらえれば分かるように，SPEA2 は，以下のファイルをコンパイルすることにより生成することができる．

oneisland.cpp
pop.cpp
gene.cpp
func.cpp
func_numerical.cpp
mutation.cpp
cross_over.cpp
select.cpp
mtrand.cpp

loadconf.cpp
main_oneisland.cpp
select_spea2.cpp
one_spea2.cpp
one_spea2_mpi.cpp
main_spea2_mpi.cpp

- NSGA-II
  以下のコマンドを実行することにより実行プログラム [nsga2] を作成することができる．

  make nsga2

  make に成功すると nsga2 という実行ファイルが生成される．これを実行すると標準の設定で NSGA-II が実行される．

  ./nsga2

  makefile を見てもらえれば分かるように，NSGA-II は，以下のファイルをコンパイルすることにより生成することができる．

  oneisland.cpp
  pop.cpp
  gene.cpp
  func.cpp
  func_numerical.cpp
  mutation.cpp
  cross_over.cpp
  select.cpp
  mtrand.cpp
  loadconf.cpp
  main_oneisland.cpp
  select_nsga2.cpp
  one_nsga2.cpp
  main_nsga2.cpp

- MPI_NSGA-II
  以下のコマンドを実行することにより実行プログラム [mpi_nsga2] を作成することができる．

  make mpi_nsga2

  make に成功すると mpi_nsga2 という実行ファイルが生成される．これを実行すると

標準の設定で NSGA-II が実行される．

mpirun –np 2 mpi_nsga2

makefile を見てもらえれば分かるように，NSGA-II は，以下のファイルをコンパイルすることにより生成することができる．

oneisland.cpp
pop.cpp
gene.cpp
func.cpp
func_numerical.cpp
mutation.cpp
cross_over.cpp
select.cpp
mtrand.cpp
loadconf.cpp
main_oneisland.cpp
select_nsga2.cpp
one_nsga2.cpp
one_nsga2_mpi.cpp
main_nsga2_mpi.cpp

- NCGA

以下のコマンドを実行することにより実行プログラム [ncga] を作成することができる．

make ncga

make に成功すると ncga という実行ファイルが生成される．これを実行すると標準の設定で NCGA が実行される．

./ncga

makefile を見てもらえれば分かるように，NCGA は，以下のファイルをコンパイルすることにより生成することができる．

oneisland.cpp
pop.cpp
gene.cpp
func.cpp
func_numerical.cpp
mutation.cpp
cross_over.cpp

select.cpp
mtrand.cpp
loadconf.cpp
select_spea2.cpp
ncga.cpp
main_ncga.cpp

・ MPI_NCGA
以下のコマンドを実行することにより実行プログラム [mpi_ncga] を作成することができる.

make mpi_ncga

make に成功すると ncga という実行ファイルが生成される．これを実行すると標準の設定で NCGA が実行される.

mpirun –np 2 mpi_ncga

makefile を見てもらえれば分かるように，NCGA は，以下のファイルをコンパイルすることにより生成することができる.

oneisland.cpp
pop.cpp
gene.cpp
func.cpp
func_numerical.cpp
mutation.cpp
cross_over.cpp
select.cpp
mtrand.cpp
loadconf.cpp
ncga.cpp
ncga_mpi.cpp
main_ncga_mpi.cpp

実行環境

2. . gcc version 2.95.4 20010319 (Debian prerelease)
debian linux (kernel 2.2.18pre21)

コンパイルオプション

2. . (無し)

2. . -O2 –funroll-loops
2. . -O6 –mcpu=pentiumpro –ffast-math –funroll-all-loops

2.3 パラメータについて

一般的な GA においては以下のようなパラメータが存在する[1].

母集団サイズ（Population size）
最大世代数（Maximum generation）
交叉率（Crossover rate）
交叉法（Crossover method）
突然変異率（Mutation rate）
突然変異法 (mutation method)

また，各種法には以下のような固有のパラメータが存在する.

[MOGA]

選択手法（Selection method）
シェアリングパラメータ (sharing need num)

[NSGA-II]

NSGA-II における選択コントロールパラメータ r (0<r<1)

GA の解探索能力はこれらのパラメータによって変化することが知られているが，一般のユーザが任意の問題について最適なパラメータ設定を行うことは困難である．本プログラムでは，これらのパラメータのデフォルト値を以下のように設定した.

[全手法共通]

母集団サイズ（100）
対象とする問題の番号 (1)
最大世代数　（250）
交叉率（1.0）
交叉法（1 点交叉）
突然変異率（1/染色体長）
突然変異法 (ビット反転)
試行回数 (1)
終了世代 (250)
評価回数の上限 (無限)
ファイルへの出力レベル (1，最低レベル)

設計変数の数 (10)
目的関数の数 (2)
個体初期化に設定ファイルを用いるか否か(0 ，用いない)

[MOGA]

選択手法（ランキングに基づくルーレット選択 , 選択手法番号 1）
シェアリングパラメータ (100)

[NSGA-II]

NSGA-II における選択コントロールパラメータ r (0.0)

これらの値は複数のテスト関数における大規模な数値実験の結果から，任意の対象問題に対して良好な性能を示すと推定された値である．なお，パラメータの値は実行時に設定ファイルを用いて変更することができる．

2.4 設定ファイルを用いたパラメータの変更

本プログラムでは，全ての手法において設定ファイルを用いてパラメータを設定することができる．

2.4.1 設定ファイルの書式

データを読み込ませるための設定ファイルの書式について説明する．設定ファイルでは，[valuename(変数名) = value(値)]のように，変数の設定を定義する．変数名はあらかじめ以下のように定義されている．

変数名	意味
pop_num	個体数(初期個体数)
func_number	対象とする問題の番号
cross_method	交叉手法
mut_method	突然変異手法
select_method	選択手法
cross_r	交叉率
mut_r	突然変異率
sharing_need_pop	シェアリングパラメータ
loop_num	試行回数(繰り返し数)
end_gene	終了世代
limit_func_count	評価回数の上限
print_level	ファイルへの出力レベル

plan_num	設計変数の数
obj_num	目的関数の数
R	NSGA-II における選択パラメータ
file_in	個体初期化に設定ファイルを用いるか否か

上記の変数を実際に設定した例をサンプルとして mof_ga.txt に示す．

```
#-----GA parameter
func_number = 10
select_method = 1
mut_r = 0.01
loop_num = 2
end_gene = 250
print_level = 1
plan_num = 10
obj_num = 2
cross_method = 2
;end
```

Mof_ga.txt

上記のように設定ファイルは，全てのパラメータを設定する必要はない．デフォルトの値で構わない場合には，省略することができる．また，" = "（イコール）の前後や変数の前にスペースを入れることも可能である．
コメントアウトするには，"#"（シャープ）または";"（セミコロン）を使用する．本プログラムでは，これら(# ;)以降の文字は読み飛ばされる仕組みとなっている．

また，本プログラムでは各手法ごとに異なる設定ファイル名を指定している．手法と設定ファイル名の関連を以下に示す．

MOGA および MPI_MOGA ： mof_ga.txt
SPEA2 および MPI_SPEA2 ： mof_spea2.txt
NSGA-II および MPI_NSGA-II ： mof_nsga2.txt
NCGA および MPI_NCGA ： mof_ga.txt

ここで．設定パラメータの持つ意味とパラメータの種類についてもう少し詳しく述べる．

[各種法の共通]

pop_num ：

母集団サイズを決定するパラメータ．基本的に 2 以上であればいくらでも構わない

func_number：

対象とする問題のパラメータ．詳細は 2.4.3 節のテスト関数を参照．

cross_method：

交叉方法に関するパラメータ．” = 1” で一点交叉．” = 2”で 2 点交叉を行う．

mut_method：

突然変異手法に関するパラメータ．本プログラムでは，ビット反転以外を実装していないため直接的な関係は無い．

cross_r：

交叉率に関するパラメータ．0.0 から 1.0 の間で設定する．

mut_r：

突然変異率に関するパラメータ．0.0 から 1.0 の間で設定する．

loop_num：

試行回数に関するパラメータ．1 以上の値を設定する．

end_gene：

終了世代に関するパラメータ．1 以上の値を設定する．

limit_func_count：

評価回数の上限を定めるパラメータ．1 以上の値を設定する．

print_level：

ファイルへの出力レベルに関するパラメータ．1 から 3 までを設定することができ，値が大きくなるほど，ファイルへの出力がより詳細なものへと変化する．

plan_num：

問題によって，設計変数の数を設定することができる．問題に依存するが，基本的に 2 以上．

obj_num：

問題によって，目的関数の数を設定することができる．基本的に 2 以上に設定する．

file_in:

個体初期化において,設定ファイル(design_variable.ini ファイル)を用いることができる．この点については 2.4.2 節の「入力ファイルを用いた初期化」で説明する．

[MOGA, MPI_MOGA に関するパラメータ]

select_method：

選択手法を設定するパラメータ．

select_method = 1: Fonseca のランキングに基づくルーレット選択

select_method = 2: Fonseca のランキングと目的関数空間でのシェアリングに基づくルーレット選択.

select_method = 3: Fonseca のランキングと目的関数空間でのシェアリングに基づくトーナメント選択 (トーナメントサイズは，使用個体数分．つまり最大) ．

select_method = 4: Fonseca のランキングと目的関数空間でのシェアリングに基づく 2 個体トーナメント選択 (トーナメントサイズは 2).

sharing_need_pop：

シェアリングに関するパラメータ．シェアリング半径(ρ_share)を決定するためのパラメータ．一般に使用個体数以下の値を設定する.

[NSGA-II , MPI_NSGA-II に関するパラメータ]

r:

NSGA-II における選択コントロールパラメータ．必ず 0 以上 1 未満を設定する．1.0 を含めた 1 以上の値を設定した場合，強制的に r=0.9 となる仕組みとなっている.

2.4.2 入力ファイルを用いた初期化

全ての手法において設定ファイルを用いて初期個体を指定することができる.

初期個体設定ファイル名は,「**design_variable.ini**」である(変更したい場合には, oneisland.h ファイルにおける「INITIAL_POPULATION_FILE」の定義を書き直す必要がある).

また，初期個体設定ファイルを用いる場合には，mof_ga.txt などのパラメータ設定ファイルにおけるパラメータ「file_in」を「1」に設定する必要がある.

設定ファイル(design_variable.ini)の例を以下に示す.

```
RunCounter    x1     x2     x3     x4     x5
1      -7.501370015  -8.850767853  -6.988393869  4.17963018    6.309546484
2      6.654656179   1.392214229   1.444718664   -3.603129042  3.797540378
3      3.550406736   -1.118594598  2.584324095   -3.938642765  -0.30946518
4      -0.538060596  6.945126972   -7.551823214  4.643517926   4.635494333
5      9.292352094   -6.869138526  2.147970777   7.330347315   1.373406009
6      10     10     10     10     10
```

design_variable.ini

上記の例のように，列が各変数を、行が各評価ケースを表し、１行で１つの個体初期化を表現する．**また，第一行目および，第一列目には必ず行名札と列名札を書く必要がある.**

本プログラムでは第一行目および，第一列目は入力値として読み込まない仕組みとなって

いる．

実際の利用の場面では、個体数とファイルのデータ行数は必ずしも一致しない場合も考えられる．本プログラムでは，その場合には

１）初期化ファイル中のデータの方が多い

→　ファイル中データの先頭ｎ行を使用

２）個体数の方が多い

→　ファイルのデータでｎ個体目まで初期化した後、

n+1 個体以降は、乱数で初期化

のように初期化を行う．

＃注意： ただし，設定ファイル(design_variable.ini)に書いた設計変数の数と対象とした問題の設計変数の数は必ず統一しておく必要がある．プログラムの方から設定ファイルにおける設計変数の数と実際の対象としている問題の設計変数の数との整合性についてチェックは一切行っていない．

2.4.3 ファイル出力

本プログラムでは，合計 7 つのファイルを出力する．以下に出力ファイルの名前と出力内容について示す．

出力ファイル名	ファイル内容
output_base_parameter_XX_YY.txt	使用したパラメータ関連の出力
output_time_XX_YY.txt	全計算時間，一試行辺りの計算時間などの出力
output_object_value_XX_YY.txt	得られた解の評価値の絶対値を出力
output_one_XX_YY.txt	得られた解の評価値をそのまま出力
output_one_plan_ XX_YY.txt	得られた解の設計変数値を出力
output_chromo_XX_YY.txt	得られた解の染色体情報を出力
output_error_XX_YY.txt	真のパレート解との誤差の履歴を出力

上記の各出力ファイル名における“ XX ”は手法名を，YY は対象とした関数番号を表している．

本プログラムでは，これらの出力ファイルの程度を“ print_level ”という変数を用いて指定することができる．以下に“ print_level ”とファイル出力への関係を示す．

print_level = 1: パラメータ関連の出力，計算時間関連の出力，最終世代における得られた解の評価値の出力

print_level = 2: print_level = 1 に加えて，最終世代における得られた解の評

価値を出力，真のパレート解との誤差の履歴を出力，各世代毎に得られた解の評価値の出力

print_level = 3: print_level = 2 に加えて，得られた解の染色体情報の出力.

2.4.4 テスト関数

本プログラムでは，計 13 のテスト関数を実装した．対象とする問題とそれを決定するパラメータ "func_number"の関係を以下に示す．

func_number = 1: 玉置の例題 2[8]

$$f_1 = \frac{1}{4}{x_1}^2$$
$$f_2 = x_1(1 - x_2) + 5$$

func_number = 2: 玉置の例題 3[8]

$$f_1 = {x_1}^2 - x_2$$
$$f_2 = -\frac{1}{2}{x_1}^2 - x_2 - 1$$

func_number = 3: 玉置の例題 4[8]

$$f_1 = -2x_1 + x_2$$
$$f_2 = x_2$$

func_number = 4: Veldhuizen の 3 目的関数[11]

$$f_1 = \frac{1}{2}({x_1}^2 + {x_2}^2) + \sin({x_1}^2 + {x_2}^2)$$
$$f_2 = \frac{(3x_1 - 2x_2 + 4)^2}{8} + \frac{(x_1 - x_2 + 1)^2}{27} + 15$$
$$f_3 = \frac{1}{{x_1}^2 + {x_2}^2 + 1} - \frac{11}{10}\exp(-{x_1}^2 - {x_2}^2)$$

func_number = 5: 玉置の例題 1[8]

$$f_1 = 2\sqrt{x_1}$$
$$f_2 = x_1(1 - x_2) + 5$$

func_number = 6: Deb の考案した多峰性のある問題[9]

$$f_1 = x_1$$
$$f_2 = g \times h$$
$$g = 1 + 10(N-1) + \sum_{i=2}^{N} x_i^{\,2} - 10\cos(2\pi x_i)$$
$$h = \begin{cases} 1 - \left(\dfrac{f_1}{g}\right)^{0.5}, & if \quad f_1 \le g, \\ 0, & otherwise \end{cases}$$

func_number = 7: Deb の考案した偏重パレートフロント[9]

$$f_1 = 1 - \exp(-4x_1)\sin^6(5\pi x_1)$$
$$f_2 = g \times h$$
$$g = 1 + 10\left(\frac{\sum_{i=2}^{N} x_i}{N-1}\right)^{0.25}$$
$$h = \begin{cases} 1 - \left(\dfrac{f_1}{g}\right)^{2}, & if \quad f_1 \le g, \\ 0, & otherwise \end{cases}$$

func_number = 8: Deb の考案した凹面多目的問題[9]

$$f_1 = x_1$$
$$f_2 = g \times h$$
$$g = 1 + 10\frac{\sum_{i=2}^{N} x_i}{N-1}$$
$$h = \begin{cases} 1 - \left(\dfrac{f_1}{g}\right)^{\alpha(\alpha=2)}, & if \quad f_1 \le g, \\ 0, & otherwise \end{cases}$$

func_number = 9: Deb の考案した不連続多目的問題[9]

$$f_1 = x_1$$
$$f_2 = g \times h$$
$$g = 1 + 10\frac{\sum_{i=2}^{N} x_i}{N-1}$$
$$h = 1 - \left(\frac{f_1}{g}\right)^{0.25} - \frac{f_1}{g}\sin(10\pi f_1)$$

func_number = 10:ZDT4[10]

$$f_1 = x_1$$
$$f_2 = g \times h$$
$$g = 1 + 10(N-1) + \sum_{i=2}^{N} x_i^{\,2} - 10\cos(4\pi x_i)$$
$$h = 1 - \left(\frac{f_1}{g} \right)^{0.5}$$

func_number = 11:ZDT6[10]

$$f_1 = 1 - \exp(-4x_1)\sin^6(5\pi x_1)$$
$$f_2 = g \times h$$
$$g = 1 + 9\left(\frac{\sum_{i=2}^{N} x_i}{N-1} \right)^{0.25}$$
$$h = 1 - \left(\frac{f_1}{g} \right)^2$$

func_number = 12:SPH-m[3]

$$f_j = \sum_{1 \le i \le n, i \ne j}^{N} (x_i)^2 + (x_j - 1)^2$$

func_number = 13:KUR[3,12]

$$f_1 = \sum_{i=1}^{N-1} (-10\exp(-0.2\sqrt{x_i^{\,2} + x_{i+1}^{\,2}}$$
$$f_2 = \sum_{i=1}^{N} (|x_i|^{0.8} + \sin^3(x_i))$$

※尚，いずれも各変数は 20bit とした.

3.アルゴリズムの説明

本章では，各アルゴリズムの簡単な解説と NCGA の詳細なアルゴリズムの仕組み，特徴について述べる．

3.1 MOGA

MOGA は，パレート的アプローチ手法（個体間の優越度合いを何らかの形で適合度に反映させる方法)における最もシンプルな手法であるといえる． パレート的アプローチを用いた GA における探索概念図を以下に示す．

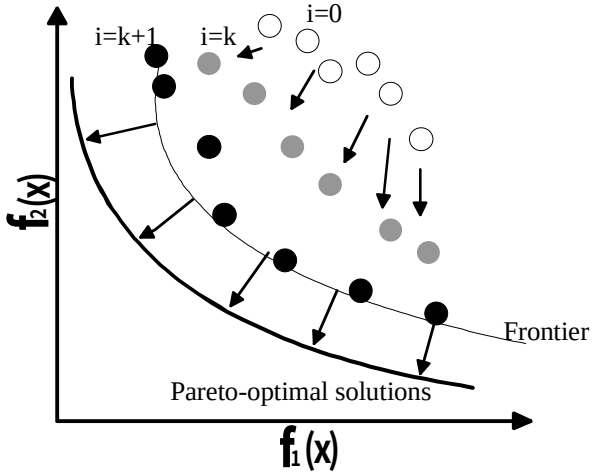


図 1：The conceptual diagram of the MOPs genetic algorithm search

MOGA における特徴を以下に示す．

- ランキングに基づく適合度割付を行う
- 探索個体群の多様性確保のために

MOGA に関する詳細は，参考文献[2]に詳しい．

3.2 SPEA2

SPEA2 は，99 年に Zitler が考案した SPEA を改良した手法であり，次のような特徴を持っている．

- 探索途中でのパレート個体の保存
- 独自の適合度割り当てを使用
- 独自の保存個体群の削減方法を利用

SPEA2 は, 2001 年度に提案された非常に新しいアルゴリズムであるがこの分野における影響力は，非常に大きく，現在，最も優れたアルゴリズムの１つといえる．
ここで，SPEA2 における適合度割り当て方法と保存個体群の削減方法について説明する．

SPEA2 における適合度割り当て

SPEA2 における適合度割り当ての概念図を以下に示す.

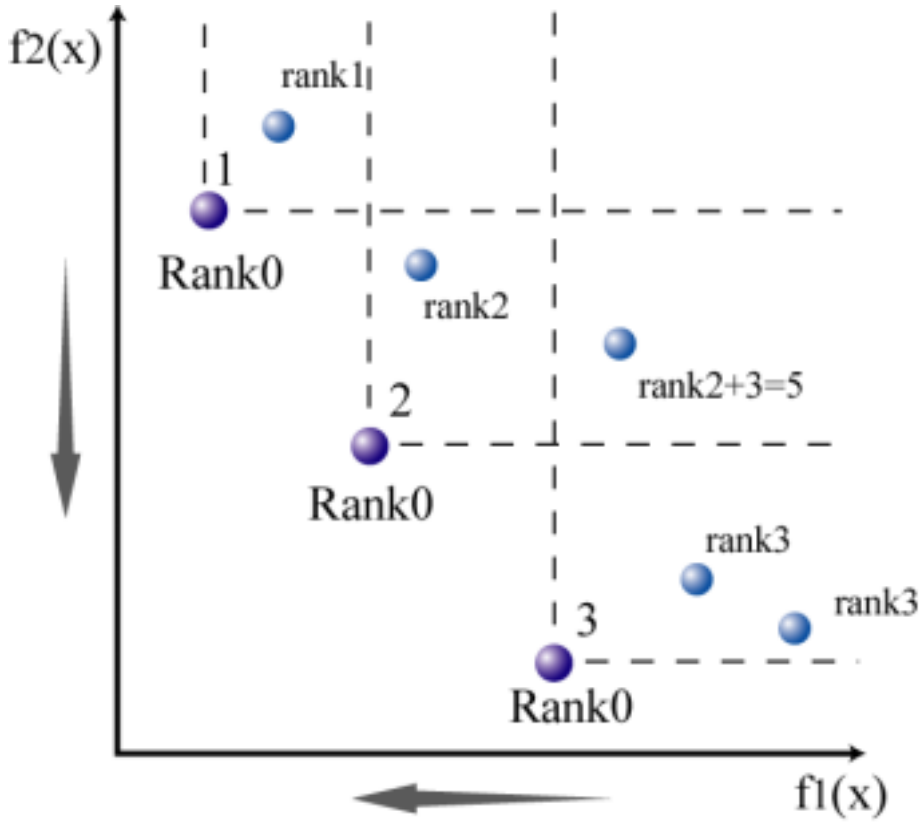


図 2：SPEA2 における適合度割付の概念図

SPEA2 では以下の流れにそって個体のランク付けを行っている.

1. 全ての個体に対して，その個体がどれだけの個体を優越しているのかその個数を調べる．ここで，ある個体 x が n 個の個体を優越している場合,「個体 x の持つ優越個体個数は，n 個である」と定義する.
2. 自分を優越している個体の持っている優越個体個数を全ての合計し，その値を自分のランク値とする.

そのため, パレート個体のランク値は 0 となり, より多くの個体に優越されている個体は, 非常に高いランク値が与えられることになる．つまり，ランク値が

この手法は，単純に解の探索具合だけでなく，密集度の高い個体群ではより悪いランク値が与えられるようになっているところが特徴である.

SPEA2 における保存個体群の削減方法

SPEA2 では, 探索途中における優良な個体を優良個体群として保存しながら探索を進める. しかし，ランク 1 個体の探索固体中に占める割合が多くなるにつれ全ての優良な個体を保

存することは不可能となるため，何らかの選択手法が必要となる．つまり，ランク 1 個体ばかりの個体群における選択方法である．
SPEA2 ではこの点について以下の図に示すようなアーカイブ端切手法(truncation method)を用いている．

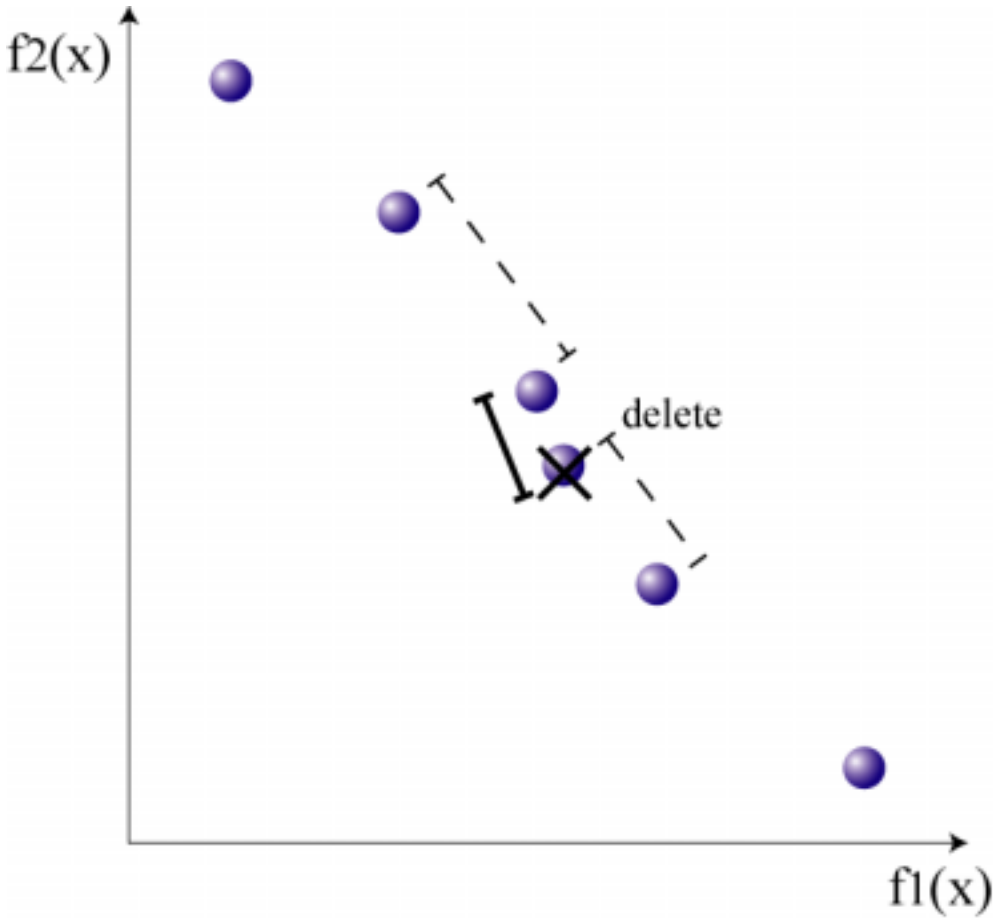


図 3：SPEA2 のアーカイブ端切手法(truncation method)

アーカイブ端切手法の仕組みを以下説明する．

1. 最も近接関係にある個体間の距離を測定し，その中で最も短い個体間距離を持っている個体のペアを選択する (最短の個体間距離を持っている個体は必ずペアになる)．
2. 選択されたペアにおいて，ペア以外の個体との個体距離を測定し，短い方の個体間距離を持っている個体を削減する．

図においては，真ん中に位置する 2 個体のペアがまず選択され，その上でペア以外の個体との個体距離が短い下の個体が削減対象に選ばれている様子を示している．この手法の利点は，必ず端の個体が選択される点である．

SPEA2 に関する詳細は，参考文献[3]に詳しい．

3.3 NSGA-II

NSGA-II は，94 年に Deb らにより考案された NSGA を改良した手法であり，次のような特徴を持っている．

- 探索途中でのパレート個体の保存
- 個体のランク付けは，ゴールドバーグにより考案された方法を使用
- 混雑度(crowding distance)の導入

NSGA-II は，2001 年度に提案された非常に新しいアルゴリズムであるがこの分野における影響力は，非常に大きく，現在，SPEA2 と並び最も優れたアルゴリズムの 1 つといえる．

3.4 NCGA

NCGA は，我々が新たに提案する新しい多目的 GA のアルゴリズムであり，以下のような特徴を持っている．

- 探索途中でのパレート個体の保存
- 局所的探索を行う遺伝的操作
- 並列処理
- SPEA2 における選択手法の利用

上記の中でも，特に局所的探索を行う遺伝的操作と並列処理は，これまでのアルゴリズムには取り入れられていない特徴であり NCGA オリジナルのものである．

NCGA は基本的に既存モデルの探索に効果的と思われる部分を多く取り込んだ手法となっている．NCGA の簡単な流れを以下示す．

step1.　初期個体を生成する．世代 $t=1$ とする．各個体の評価をスレーブにて行い，これらの初期個体群をアーカイブ個体群(A_t)とする．

step2.　アーカイブ個体群(A_t)を探索個体群(P_t)にコピーし，P_t を 1 つの目的関数値を基準にソートし並び替える． この際，着目する目的関数は毎世代ごとに変更する．

step3.　探索個体群(P_t)を順に非復元抽出し 2 個体のペア個体群を選択する．

step4.　選択された 2 個体を用いて交叉，突然変異，評価を行い，新たに得られた 2 個体を選択した 2 個体のペアと入れ替える．全ての個体が選択されるまで Step3, Step4 を繰り返す．この結果，探索個体群が全て更新される(P_{t+1})．

step5.　探索個体群(P_{t+1})とアーカイブ個体群(A_t)との比較を行い，アーカイブ個体群を更新する(A_{t+1})．この際，アーカイブ更新の方法として SPEA2 の手法を用いた．つまり，本手法では SPEA2 における適合度割り当て，アーカイブ更新が用いられている．

step6.　また，SPEA2 などと同様，アーカイブ個体群はあらかじめ設定した個体数N個分の個体を優良個体として保存している．

step7.　終了条件を満たすかどうか判定を行う．終了条件を満たせば終了，満たさない場合には，世代 ***t=t+1*** を行い，Step2 へ戻る．

このように提案する NCGA は, 個体ペアを選択する前に探索個体群を任意の目的関数軸を基準にソートし並び替えることにより，近傍交叉を実現している．ただし，ここでのソートを常にある一定の目的関数軸を基準に行うと繰り返し同じペア同士での交叉になる恐れがあるため以下のような方法を用いている．

- 毎世代毎に基準となる目的関数値を変化
- 個体のソート後，ソート対象となっている個体の数の 1 割程度の範囲において近傍シャッフルを実行

上記における近傍シャッフルとは，ある一定の範囲内で個体をランダムに並び替えるものである．例えば，100 個体の個体群が対象である場合，各個体は幅最大 10 の範囲で乱数を用いたシャッフルが行われる．そのため，個体ペアは毎世代ごと変化することになる．さらに，上記の近傍シャッフルを行うことにより，各個体間の情報交換も活発化されるものと思われる．

また，パレート個体群(A_t)の更新には SPEA2 における適合度割り当ておよび環境選択(Environmental selection)を用いている．一方で，SPEA2，NSGA-II などの手法が用いているパレート個体群(A_t)から探索個体群(P_t)を選択するマッティング選択(Mating Selection)を行っていない．これは，同じ個体が複数存在すると近傍交叉に悪影響を及ぼす可能性があるからである．そのため，NCGA ではパレート個体群(A_t)のコピーを探索個体群(A_t)として用いている．

また，NCGA はマスタースレーブ型並列モデルに容易にインプリメント可能である．すなわち，NCGA の各手順のうち Step4 をスレーブノードで行い，それ以外の操作をマスターで行うというモデルである．

NCGA におけるマスタースレーブ型並列モデルでは，基本的な役割がマスターノードとスレーブノードによって完全に分担されている．特に，従来のマスターノード型並列モデルと異なり，評価だけでなく交叉，突然変異といった選択以外の遺伝的操作をスレーブノードが行うという特徴を持っている．このことによって，従来までのマスターノード型並列モデルにおけるマスターノードの高負荷を緩和することができる．マスタースレーブモデルへ適用した場合の NCGA の概念図を以下に示す．

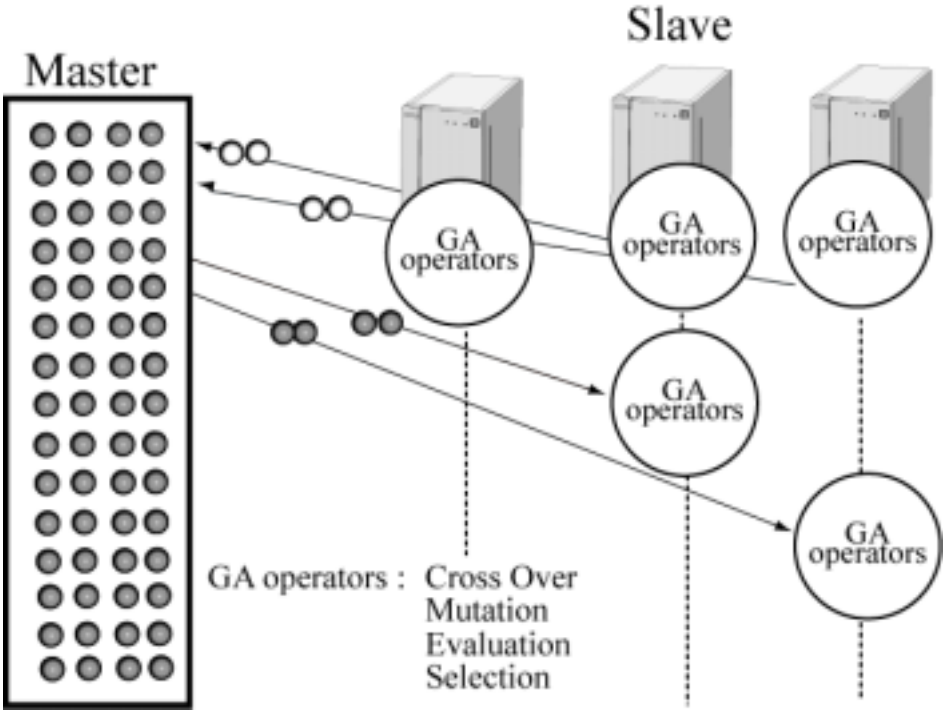


図 4：NCGA におけるマスタースレーブモデルの概念図

4.性能の評価

本章では，MOGA[2]，SPEA2[3]，NSGA-II [4,5]，NCGA[6]の性能比較の結果について示す．本プログラムでは，テスト関数として 13 の関数が用意されている．ここでは，そのうち ZDT4(func_number = 10)，KUR(func_number = 13)についての性能比較について述べる．

4.1 問題設定

本実験では，試行回数以外の部分は全てデフォルト値を用いた．用いたパラメータについて以下示す．

変数名	値
pop_num	100
cross_method	1 (1 点交叉)
mut_method	1 (ビット反転)
cross_r	1.0
mut_r	遺伝子長分の 1
loop_num	10
end_gene	250

MOGA 用パラメータ

変数名	値
select_method	1 (ランキングに基づくルーレット選択)
sharing_need_pop	100

NSGA-II 用パラメータ

変数名	値
R	0.0

また，対象とした問題は次の 3 種類の問題である．

- ZDT4(10 設計変数)
- KUR (10 設計変数)
- KUR (100 設計変数)

3.2 結果

10 試行計算により得られた全てのパレート解を問題ごとにプロットした図を図.1 ～ 3 に示す．尚，全ての問題は 2 目的の最小化問題である．そのため，パレート解は両目的の最小値の方向，すなわち左下にあるほど良好な解であるといえる．

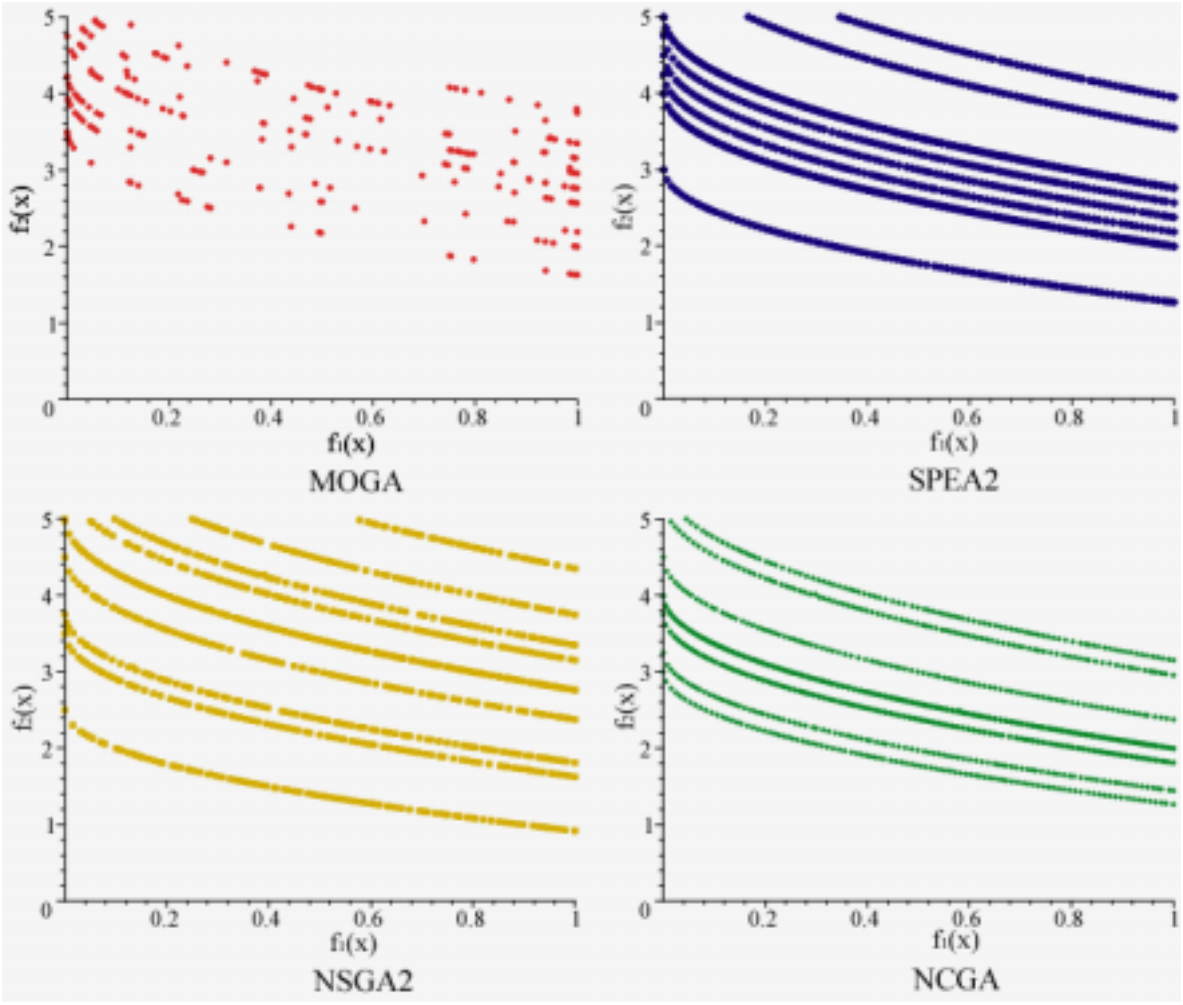


図 5：ZDT4(10 設計変数)の結果

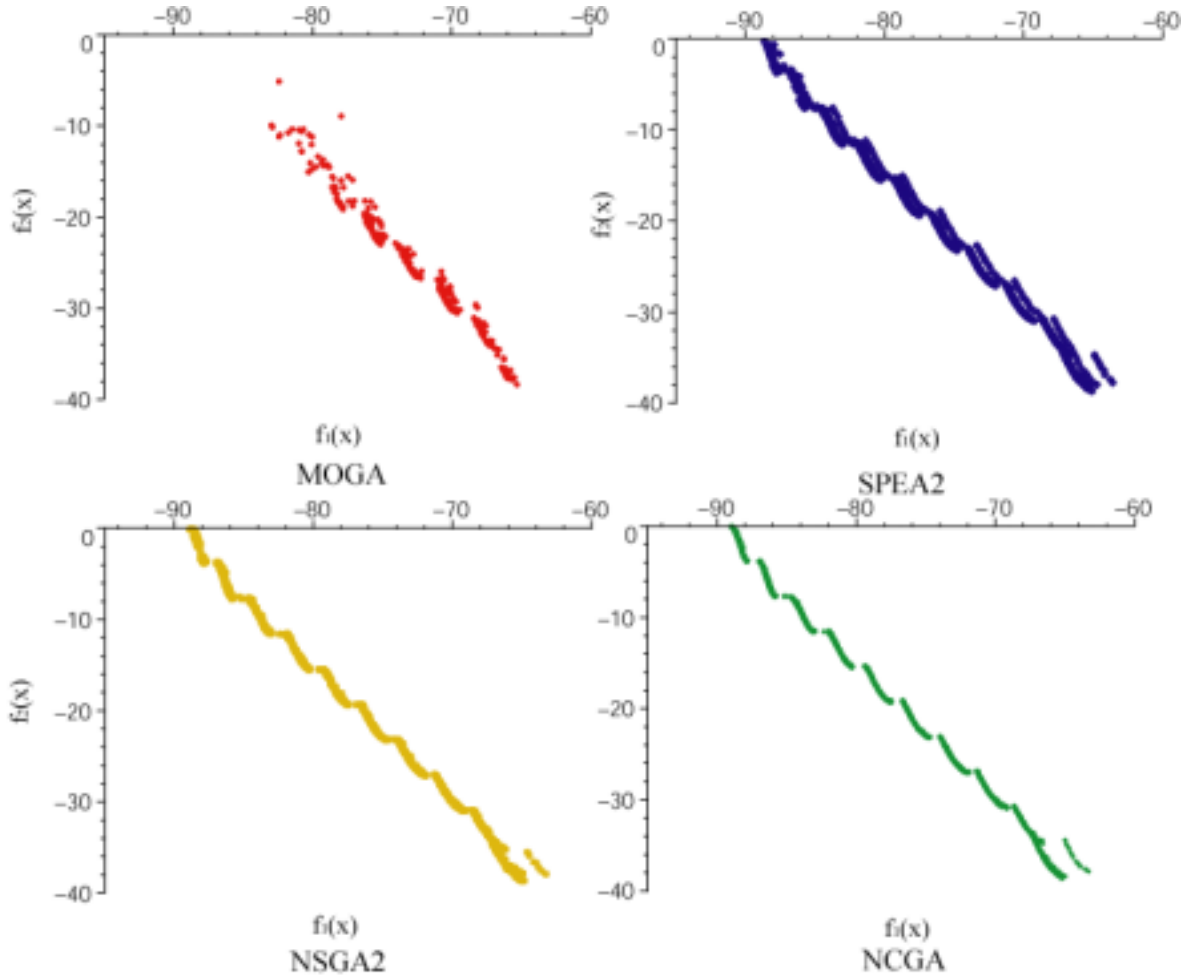


図 6：KUR の結果（10 設計変数）

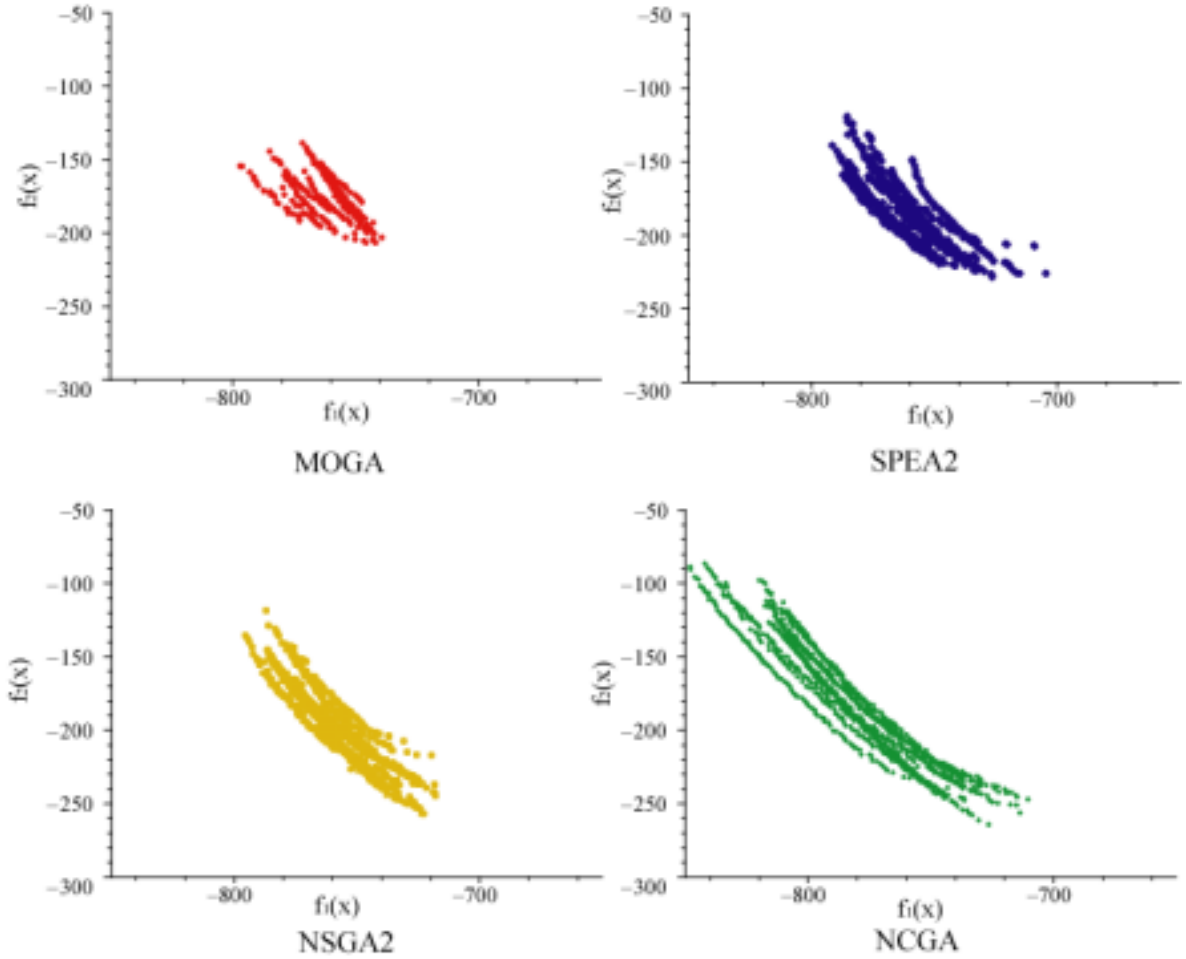


図 7：KUR の結果（100 設計変数）

これらの結果より，MOGA がその他の手法に比べて大きく劣っている様子が分かる．これは，MOGA のみがパレート保存戦略を用いておらず，良好な個体が探索途中を失っているためであると考えられる．また，MOGA 以外の 3 手法を比較した場合，それほど優位な差は見られないことが分かる．

しかし，100 設計変数の問題では NCGA が SPEA2，NSGA-II に比べて明確な優位性を示している事が分かる．

これらの結果から，MOGA,SPEA2，NSGA-II ，NCGA の 4 手法の内で NCGA は，最も良好な結果を得ることができ，逆に MOGA は他の手法に比べあまり良好な結果を得ることができないことが分かった．

5．プログラム詳説

本章では，実装されたプログラムについて解説する． 本プログラムは，MOGA，SPEA2，NSGA-II ，NCGA の 4 種類の手法に対応しており，それぞれの手法についてシングルバージョン，並列バージョンの 2 種類が用意されている． そのため，計 8 種類の手法を実装することが可能となっている．

本プログラムは，全 27 ファイルから成り立っている．以下，ファイルとそのファイルの中身について簡単に示す．

ファイル名	内容
cross_over.cpp	交叉に関するファイル
func.cpp	評価関数に関するファイル
func_numerical.cpp	連続関数テスト問題(評価関数)に関するファイル
gene.cpp	遺伝子に関するファイル
loadconf.cpp	設定ファイルからの読み込みに関するファイル
main_m_s.cpp	MOGA のマスタースレーブ型並列モデルに関する main()関数
main_ncga.cpp	NCGA に関する main()関数
main_ncga_mpi.cpp	NCGA のマスタースレーブ型並列モデルに関する main()関数
main_nsga2.cpp	NSGA-II に関する main()関数
main_nsga2_mpi.cpp	NSGA-II のマスタースレーブ型並列モデルに関する main()関数
main_oneisland.cpp	MOGA に関する main()関数
main_spea2.cpp	SPEA2 に関する main()関数
main_spea2_mpi.cpp	SPEA2 のマスタースレーブ型並列モデルに関する main()関数
mtrand.cpp	乱数発生に関するファイル
mutation.cpp	突然変異に関するファイル
ncga.cpp	NCGA に関するファイル
ncga_mpi.cpp	NCGA のマスタースレーブ型並列モデルに関するファイル
one_mpi.cpp	MOGA のマスタースレーブ型並列モデルに関するファイル
one_nsga2.cpp	NSGA-II に関するファイル
one_nsga2_mpi.cpp	NSGA-II のマスタースレーブ型並列モデルに関するファイル
one_spea2.cpp	SPEA2 に関するファイル
one_spea2_mpi.cpp	SPEA2 のマスタースレーブ型並列モデルに関するファイル
oneisland.cpp	GA 操作全体に関するファイル
pop.cpp	母集団に関するファイル

select.cpp	選択に関するファイル
select_nsga2.cpp	NSGA-II における選択手法に関するファイル
select_spea2.cpp	SPEA2 における選択手法に関するファイル

上記のファイルの内，全ての手法に共通する最もコアなファイルは，以下に示す 10 ファイルである．

cross_over.cpp
func.cpp
func_numerical.cpp
gene.cpp
loadconf.cpp
mtrand.cpp
mutation.cpp
oneisland.cpp
pop.cpp
select.cpp

上記の 10 ファイルには，13 のクラスが記述されており，これら 13 のクラスが多目的遺伝的アルゴリズムにおける主要な役割を果たしている．以下では MOGA，NCGA を例にとってクラスとファイルの関係および各クラスの大まかな中身について説明する．

5.1 ファイルとクラスの関係

5.1.1 MOGA

本節では，MOGA において用いているファイルとそのファイルに含まれるクラスについて解説する．

[main_oneisland.cpp ファイル]

- MOGA の main 関数

[oneisland.cpp ファイル]

- クラス名

  F_out　：ファイル出力に関するクラス（One クラスに含まれる）
  One　　：GA 操作に関するクラス（全てのクラスを含む）

[pop.cpp]

- クラス名

  Pop　：母集団に関するクラス（One クラスに含まれる）

[gene.cpp]

- クラス名

  Chromo ：染色体に関するクラス（Gene クラスに含まれる）

  Gene　：個体に関するクラス （Pop クラスに含まれる）

  Obj　：個体の評価値，適合度値に関するクラス（Gene クラスに含まれる）

[func.cpp]

- クラス名

  Func　：目的関数に関するクラス（One クラスに含まれる）

[func_numerical.cpp]

- クラス名

  Numerical_func（Func クラスの派生クラス）

  ：数学的テスト関数による評価関数に関するクラス

[crossover.cpp]

- クラス名

  Cross　：交叉に関するクラス（One クラスに含まれる）

[mutation.cpp]

- クラス名

  Mutation

  突然変異に関するクラス（One クラスに含まれる）

[select.cpp]

- クラス名

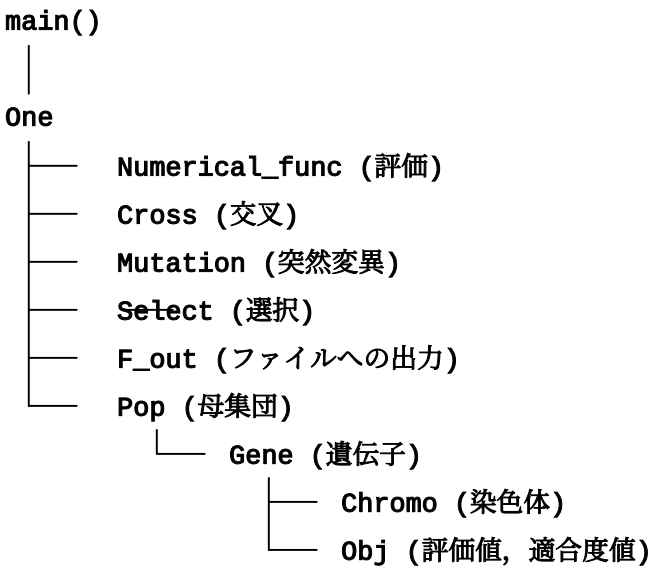
  Select ：選択に関するクラス（One クラスに含まれる）

[mtrand.cpp]

- クラス名

  MTRand ：乱数に関するクラス（Cross クラス，Mutation クラスなどに含まれる）

MOGA におけるクラス間の相関関係について以下示す.



5.1.2 NCGA

本節では, NCGA において用いるファイルとそのファイルに含まれるクラスについて解説する.

2 章でも説明したとおり,NCGA は MOGA において用いている関数の多くを用いている.NCGA において用いるファイルの内，MOGA と重なるファイルを以下に示す.

```
oneisland.cpp
pop.cpp
gene.cpp
func.cpp
func_numerical.cpp
mutation.cpp
cross_over.cpp
select.cpp
mtrand.cpp
loadconf.cpp
```

上記で示したように，NCGA は MOGA におけるメイン関数 “ main_oneisland.cpp ” 以外の全てのファイルを利用している．一方，NCGA では次の 3 つの MOGA には無いファイルをコンパイルに用いている.

[main_ncga.cpp ファイル]

- NCGA の main 関数

[ncga.cpp]

- クラス名

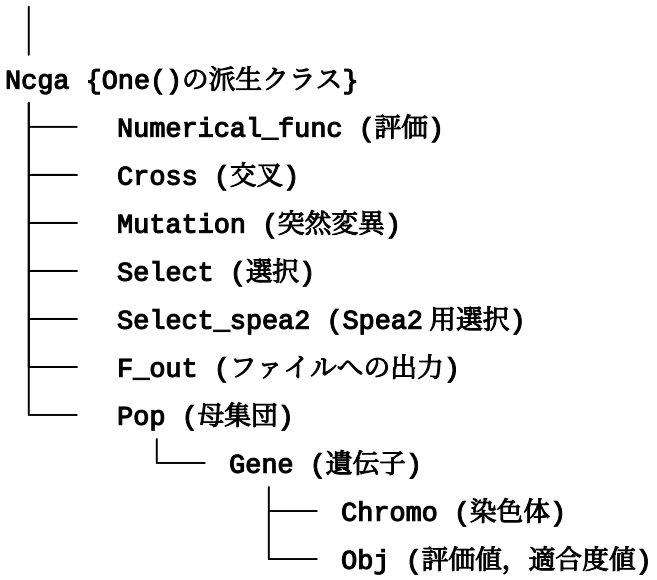
Ncga ：NCGA における遺伝的操作に関するクラス（One クラスの派生クラス）

[select_spea2.cpp]

- クラス名

  Select_spea2 ：SPEA2 に関するクラス（One_spea2, Ncga に含まれる．Select クラスの派生クラス）

NCGA におけるクラス間の相関関係について以下示す．



5.2 各クラスの説明

本節では，各クラスの詳細な中身について説明する．具体的には，各クラスのメンバ変数，メンバ関数を定義しているヘッダファイルの中身と各変数，各関数の持つ簡単な意味について述べる．

GA 操作の表現（One クラス）

One クラスは，プログラムにおける最も外枠に位置するクラスであり，母集団を用いた GA 操作を行う．そのため，母集団を表す Pop クラス，各種遺伝的操作を表す Func クラス，Cross クラス，Mutation クラス，Select クラスなどを変数として持ち合わせている．

クラス名	One	
内容	GA 操作を表現	
メンバ変数	mutable MTRand mtrand	乱数

	double avg_generation	終了世代の平均値
	int func_count	評価計算回数
	double s_time	1試行における計算開始時間
	double e_time	1試行における計算終了時間
	double all_s_time	全試行における計算開始時間
	double all_e_time	全試行における計算終了時間
	double func_time	1試行の評価計算時間
	Numerical_func func	クラス Numerical_func の変数呼び出し
	Cross cross	クラス Cross の変数呼び出し
	F_out f_out	クラス F_out の変数呼び出し
	Mutation mut	クラス Mutation の変数呼び出し
	Select select	クラスSelectの変数呼び出し
	variable_str variable	パラメータの集合
	Pop pop	クラスPopの変数呼び出し
	int generation	世代数
	FILE *fp1	すべてのパレメータをファイルへ出力
	FILE *fp2	誤差の履歴をファイルへ出力
	FILE *fp3	計算にかかった時間などをファイルへ出力
	FILE *fp4	全世代における母集団の評価値の絶対値をファイルへ出力
	FILE *fp5	最終世代におけるパレート個体の染色体情報をファイルへ出力
	FILE *fp6	設計変数値をファイルへ出力
	FILE *fp7	パレート個体の評価値をファイルへ出力

	u8 life_flag	探索終了における条件の設定値
メンバ関数	One()	コンストラクタ
	virtual ~One()	デストラクタ
	void init_pop()	初期個体を設定ファイルから**読む込まない**場合のクラス Pop の初期化
	virtual void cross_over()	交叉の呼び出し
	virtual void all_open_w_file()	すべての出力ファイルを開く
	virtual void mutation()	突然変異の呼び出し
	virtual void initGA()	母集団の初期化
	virtual void init_base()	初期化された母集団の探索
	virtual void set_parameter()	パラメータの入力
	virtual void initialize()	各試行毎の探索母集団の初期化
	void initialize(Pop &)	各試行毎の探索母集団の初期化
	void file_init_pop(Pop &dum_pop, ifstream& fin);	初期個体を設定ファイルから**読む込む**場合に用いられる母集団の初期化関数
	virtual void selection()	選択の呼び出し
	virtual void function()	関数の呼び出し
	virtual void terminal_check()	終了判定
	virtual void cycle()	試行回数分の繰り返し
	virtual void next_1_gene()	1 世代のループ
	virtual void o_file()	設定ファイルの読み込み
	virtual void set_parameter_default()	デフォルトパラメータの読み込み
	virtual void all_open_file_close()	ファイルを閉じる
	virtual void f_output()	ファイルへの出力

	void f_output(Pop&)	ファイルへの出力
	virtual void f_set_output()	パラメータ変数のファイルへの出力
	void finish_write()	全計算時間や全計算回数のファイルへの出力
	inline void set_s_time()	s_time の値の入力
	inline void set_e_time()	e_time の値の入力
	inline void set_all_s_time()	all_s_time の値の入力
	inline void set_all_e_time()	all_e_time の値の入力
	inline int get_generation()	generation の値の出力
	inline double get_e_s_time()	e_s_time の値の出力
	inline double get_all_e_s_time()	all_e_s_time の値の出力
	inline int get_select_pop_num()	select_pop_num の値の出力
	inline int get_first_pop_num()	first_pop_num の値の出力
	void non_dominated_space (Pop &dum_pop)	全個体の平均誤差の出力
場所	oneisland.h	

目的関数の表現（Func クラス）

Func クラスは，遺伝的操作における評価部分を行うためのクラスである．評価部分は，問題変更のために非常に大きく関わるため Func クラス自身の中身を NULL 状態として具体的な評価部分は，派生クラスを作成し，そこで行うような仕組みを用いた．すなわち，Func クラスは，便宜的な枠組みを示しているだけである．

本プログラムでは，Func クラスの派生クラスとして Numerical_func というクラスを作成し，実際の評価部分はそのクラスに記述するという方法を用いた．

クラス名	Func	
内容	目的関数を表現	
メンバ変数	int pop_num;	総個体数
	static int obj_num;	目的関数の数
	static int func_number;	問題の種類
	static int plan_num;	設計変数の数
	static int chromosome_length;	遺伝子長
	static int func_count;	評価計算回数
	static double sleep_time;	評価の遅延
メンバ関数	Func()	コンストラクタ
	Func(const Func&)	コンストラクタ
	virtual ~Func()	コンストラクタ
	virtual void init_Func (const int&);	目的関数の初期化
	inline int get_plan_num()	Plan_num の値を出力
	inline void set_plan_num (const int &x)	Plan_num の値を入力
	inline double get_sleep_time()	Sleep_time の値を出力
	inline void set_sleep_time (const double &x)	Sleep_time の値を入力
	inline int get_pop_num()	Pop_num の値を出力
	inline void set_pop_num (const int &x)	Pop_num の値を入力
	inline int get_obj_num()	Obj_num の値を出力
	inline void set_obj_num (int &x)	Obj_num の値を入力
	inline int get_func_number()	Func_number の値を出力
	virtual void set_func_number(int &x)	Func_number の値を入力

	inline int get_func_count()	Func_count の値を出力
	inline void re_set_func_count()	Func_count の値を初期化
	virtual void func(Pop&)	母集団の評価
	virtual void func (Gene **,const int&)	母集団の評価
	virtual void func(Gene &)	個体の評価
	virtual void set_func_count (const int &x)	マスタースレーブモデルにおけるスレーブでの func_count の値の設定
	void sleep()	スリープ （sleep_time 秒）
	void sleep(double x)	x 秒のスリープ
場所	func.h	

数学的テスト関数による評価関数（クラス Func の派生クラス，Numerical_func クラス）

本クラスは, Func クラスの派生クラスであり, 実際の評価関数の中身などが書かれている. One クラスより GA 操作における評価部分で呼び出され，母集団の評価を行っている.

クラス名	Numerical_func	
内容	数学的テスト関数による評価関数	
メンバ変数	mutable MTRand mtrand	乱数
メンバ関数	void func1(Gene&)	関数の呼び出し（func1~func13 あり）（関数 func()の子関数）
	double real_func1(Gene&)	関数の真の解との誤差を返す（real_func1,5,10~12）（関数 real_func の子関数）
	bool check1(Gene&)	問題の制約条件の判定（check1～check13 あり）（関数 check()の子関数）

	virtual double real_func(Pop &)	関数の真の解との誤差をかえす
	inline double rand()	０~１までの乱数
	inline int rand_int(int n)	最大ｎの整数の乱数
	virtual ~Numerical_func()	デストラクタ
	virtual void func(Pop&)	関数の呼び出し
	virtual void func (Gene **,const int&)	関数の呼び出し
	virtual void func(Gene &)	関数の呼び出し
	virtual bool check(Gene &)	制約条件を満たしたいるかの判定
	virtual void mv_safe(Gene&)	個体の制約条件内への引き戻し
	virtual void set_func_number (const int &x)	func_number の入力
	virtual void set_func_number (const int &x,const int &y)	func_number の入力
	virtual void set_func_number (const int &x,const int &y,const int &z)	func_number の入力
	virtual void pop_create (Gene **,const int&)	母集団の生成
	virtual void pop_create(Pop &)	母集団の生成
場所	func_numerical.h	

交叉の方法（Cross クラス）

Cross クラスは GA 操作における交叉を行うクラスであり，One クラスにより用いられている．尚，本クラスでは交叉手法として一点交叉と 2 点交叉のみを実装しており，これらはパラメータ”cross_method” により変更することができる．

Cross_method = 1: １点交叉

Cross_method = 2: 2点交叉

クラス名	Cross	
内容	交叉の方法	
メンバ変数	Gene Child[2]	子個体
	Gene Parent[2]	親個体
	Static double cross_r	交叉率
	Static int cross_method	交叉手法
	Static int plan_num	設計変数の数
	Static int child_num	子個体の数
	Static int need_num	交叉に必要な親の数
	Static int chromo_len	遺伝子長
	int pop_num	総個体数
	mutable MTRand mtrand	乱数
メンバ関数	virtual ~Cross()	デストラクタ
	Inline double genrand()	０～１の乱数
	Inline int genrand_int (int n)	最大ｎの整数の乱数
	Inline double get_cross_r()	Cross_r の値を出力
	Inline void set_cross_r(double x)	Cross_r の値を入力
	Inline int get_need_num()	Need_num の値を出力
	Inline int get_cross_method()	Cross_method の値を出力
	inline void set_cross_method(int x)	Cross_method の値を入力
	Inline int get_pop_num()	Pop_num の値を出力
	Inline void set_pop_num(int x)	Pop_num の値を入力
	inline void set_plan_num(const int &x)	Plan_num の値を入力

	inline void set_chromo_len (const int &x)	Chromo_len の値を入力
	virtual void cross_over(Pop& a)	交叉を実行（交叉後,親を残さない）
	virtual void cross_over (Gene **,int &;)	交叉を実行（交叉後,親を残さない）
	virtual void cross_method_exe (Gene **,const int &)	交叉を実行
	virtual void CMX_cross_overl (Gene **ap_gene,int &tmp_number,const int& what_times)	複数回,交叉を実行
	virtual void One_point (const int &cross_point)	１点交叉
	virtual void Two_point()	２点交叉
	virtual void random_shuffle (Gene **ap_gene,const int &tmp_number)	ランダムに並び替えをおこなった個体を交叉
場所	Cross_over.h	

突然変異の表現（Mutation クラス）

Mutation クラスは，GA 操作における突然変異を行うクラスである．One クラスにより用いられ，母集団に対して突然変異を行っている．尚，本プログラムでは突然変異手法としてビット反転にのみ対応している．

クラス名	Mutation	
内容	突然変異を表現	
メンバ変数	static int length;	遺伝子長
	static double mut_r;	突然変異率
	static int mut_method;	突然変異の手法

	static int pop_num;	総個体数
	static int plan_num;	設計変数の数
	mutable MTRand mtrand;	乱数
メンバ関数	inline double genrand()	０～１の乱数
	inline int genrand_int (const int& n)	最大ｎの整数の乱数
	inline void set_plan_num (const int &x)	plan_num の値を入力
	inline void set_mut_method (const int& x)	mut_method の値を入力
	inline int get_mut_method()	mut_method の値を出力
	inline void set_length (const int& x)	length の値を入力
	inline int get_length()	length の値を出力
	inline void set_mut_r (const double& x)	mut_r の値を入力
	inline double get_mut_r()	mut_r の値を出力
	inline int get_pop_num()	pop_num の値を出力
	inline void set_pop_num (const int& x)	pop_num の値を入力
	virtual void mut(Pop &)	突然変異の呼び出し
	virtual void mut (Gene **,const int &)	突然変異の呼び出し
	virtual void mutation_exe (Gene &gene)	突然変異の実行
	virtual void bit_reverse (Gene &a)	突然変異（ビット反転）の実行
場所	mutation.h	

選択の表現（select クラス）

選択手法に関するクラス．Select クラスでは，基本的に MOGA における選択手法にのみ対応している．MOGA 以外の手法における選択手法は，それぞれ select クラスの派生クラスにより実現されている．

また, select クラスでは選択手法に関する関数だけでなくソートに関する各種関数が定義されている.

クラス名	Select	
内容	選択を方法	
メンバ変数	static int select_method	選択の種類
	int pop_num	総個体数
	static int obj_num	目的関数の数
	static int select_pop_num	選択する個体数
	static int sharing_need_pop	シェアリングの必要個体数
	mutable MTRand mtrand	乱数
メンバ関数	inline double genrand()	0~1の乱数
	inline int normal_genrand_int (int n)	最大nの整数の乱数
	inline int get_select_method()	Select_method の値を出力
	inline void set_select_method (int x)	Select_method の値を入力
	inline int get_pop_num()	pop_num の値を出力
	inline void set_pop_num(int x)	pop_num の値を入力
	inline void set_obj_num(int x)	obj _num の値を入力
	inline int get_obj_num()	obj _num の値を出力
	inline void set_select_pop_num (int x)	select_pop_num の値を入力
	inline int get_select_pop_num()	select_pop_num の値を出力
	inline void set_sharing_need_pop (int x)	sharing_need_pop の値を入力

	inline int get_sharing_need_pop()	sharing_need_pop の値を出力
	inline void set_last_sharing_pop (int x)	last_sharing_pop の値を入力
	inline int get_last_sharing_pop()	last_sharing_pop の値を出力
	void selection(Pop &)	選択を実行
	void selection (Gene **ap_gene,int &num)	選択を実行
	void ranking_squre (Gene **ap_gene,int &num)	適合度の値を２乗
	void roulet_select(Pop &)	ルーレット選択
	void roulet_select_normal (Gene **,const int&)	ルーレット選択
	void roulet_select (Gene **,int&,const int&)	ルーレット選択
	void tournament_select(Pop &)	トーナメント選択
	void tournament_select (Gene **ap_gene,int& dum_pop_num,const int& dum_dum)	トーナメント選択
	void tournament_select_alpha (Gene **,int &)	ランク１個体以外をトーナメント選択
	void tournament_select_alpha (Gene **,int &,const int&)	ランク１個体以外をトーナメント選択
	void obj_sharing(Pop &)	目的関数によるシェアリング
	void obj_sharing (Gene **,int&,const int&)	目的関数によるシェアリング
	void ranking_con(Pop &)	ランキングによる適合度割り付け
	void ranking_con (Gene **,const int&,double **,const int&)	ランキングによる適合度割り付け

	void select_pop(Pop &)	個体の選択
	void select_pop(Gene **,int&)	個体の選択
	void sort_pop(Pop &)	母集団を評価関数"f1"を基準に最大順にソーティング.
	void f_sort(Pop &)	母集団を適合度関数値を基準に最大順にソーティング
	void new_sort(Pop &a,int b)	母集団 a を評価関数 b 番目の値を基準に最大順にソーティング
	void reduce_same_pop(Pop &)	同じ個体を削除
	void reduce_same_pop (Gene **,int&)	同じ個体を削除
	void only_pareto_select(Pop &)	パレート個体のみを選択
	void only_pareto_check (Gene **,const int&)	母集団のパレート解のみをチェックする
	void obj_setting (Pop &perfect)	母集団全体の値を用いて各個体の持つ評価値をスケーリングした値を設定し直す
	void bi_select(Pop &pop)	２個体トーナメント選択
	void bi_select_alpha (Gene **ap_gene,int& dum_pop_num)	２個体トーナメント選択
	void bi_select_alpha (Gene **ap_gene,int& dum_pop_num,const int& dum_dum)	２個体トーナメント選択
	void choice (Gene **,int & x, const int& y)	各個体について適合度の高い順に x 個選択
場所	select.h	

ファイル出力の表現（F_out クラス）

クラス名	F_out	
内容	ファイル出力を表現	
メンバ変数	static int obj_num	目的関数の数
	double *avg_obj_value	目的関数値の平均値
	double e_s_time	1 試行分の計算時間
	double func_time	評価計算時間
	double avg_func_time	全試行の評価計算時間の平均
	double avg_e_s_time	全試行の計算時間の平均
	double all_e_s_time	全試行の計算時間
	int all_func_count	総評価計算回数
	int func_count	評価計算回数
	double avg_func_count	評価計算回数の平均値
	int all_generation	総世代数
	int generation	世代数
	double avg_generation	世代の平均値
メンバ関数	F_out()	コンストラクタ
	F_out(const int&)	コンストラクタ
	F_out(const F_out&)	コンストラクタ
	~F_out()	デストラクタ
	void avg_obj(Pop& ,FILE *)	目的関数値の平均値のファイルへの出力
	void avg_obj(Pop&)	目的関数値の平均値のファイルへの出力
	void all_obj(Pop& ,FILE *)	母集団のすべての固体の評価値の絶対値をファイルに出力
	void all_obj_s (Pop& ,FILE *)	母集団のすべての固体の評価値をファイルに出力
	void f_end(Pop& ,FILE *)	母集団における各目的関数の最大値,最小値をファイルへ出力
	void gather(FILE *)	全試行分の計算時間の平均値などをファイルへ出力

	void gather_s(FILE *)	1 試行分の計算時間の平均値などをファイルへ出力
	inline void set_e_s_time (const double& x)	e_s_time の値を入力
	inline void set_obj_num (const int& x)	obj_num の値を入力
	void init(const int& x)	クラス F_out の初期化
	void all_chromo Pop& a,FILE *fp)	すべての染色体情報をファイルへ出力
	void all_plan （Pop& a,FILE *fp）	すべての設計変数の値を出力
場所	oneisland.h	

母集団の表現（Pop クラス）

クラス名	Pop	
内容	母集団全体を表現	
メンバ変数	double *max_obj_value	目的関数値の最大値
	double *min_obj_value	目的関数値の最小値
	int pop_num	総個体数
	static int obj_num	目的関数の数
	static int max_pop	母集団の最大値
	static int plan_num	設計変数の数
メンバ関数	Gene **ap_gene	クラス Gene の２重ポインタ変数
	Pop()	コンストラクタ
	Pop(const Pop&)	コンストラクタ
	Pop(const int&)	コンストラクタ
	Pop (const int&x ,const int&z)	クラス Pop の初期化 x：個体数 z：目的関数の数
	double *avg_obj	全個体の目的関数の平均値
	double& set_max_obj (const int i)	i 番目の目的間数値の最大値を入力

	double& set_min_obj (const int i)	i 番目の目的間数値の最大値を出力
	inline int get_pop_num()	Pop_num の値を出力
	inline void set_pop_num (const int& x)	Pop_num の値を入力
	inline void set_obj_num (const int& x)	Obj_num の値を入力
	inline void set_plan_num (const int& x)	Plan_num の値を入力
	inline int get_obj_num()	Obj_num の値を出力
	virtual Pop operator =(const Pop&)	演算子「＝」の定義
	virtual Pop operator =(vector<Gene>& f)	演算子「＝」の定義
	virtual Pop operator +=(const Pop&)	演算子「＋＝」の定義
	virtual void copy (const Pop&)	クラス Pop のコピー
	virtual void plus_copy (const Pop&,const int&)	指定された母集団クラスの中身を自身末尾に加える
	virtual void init_ap_gene (const int& ,const int&)	２重ポインタ変数 ap_gene の初期化
	void flag_change_safe()	flag_chang の値をすべて safe に変える
	void flag_change_out()	flag_chang の値をすべて out に変える
場所	pop.h	

個体（遺伝子）の表現（Gene クラス）

クラス名	Gene	
内容	個体を表現	
メンバ変数	int Rna_size	遺伝子のバイト数
	Vectot<double> variable_min	評価値の最小値
	Vectot<double> variable_max	評価値の最大値
	Vector<Chromo> locus	遺伝子長
	int real_chrom	遺伝子が可変のときの最大遺伝子長
	int plan_num	設計変数の数
	int obj_num	目的関数の数
	U8 safe_flag	制約条件をみたしているかどうかの判定
	U8 chang_flag	各種遺伝的操作によって個体の変更を行われたかどうかの判定
	Obj obj	Obj クラスの変数
メンバ関数	Gene()	コンストラクタ
	Gene(const int&)	コンストラクタ
	Gene(const int& ,const int&)	コンストラクタ
	Gene(const Gene&)	コンストラクタ
	virtual ~Gene()	デストラクタ
	Gene init (const int&, const int&)	遺伝子の初期化
	Void copy(const Gene&)	遺伝子のコピー
	inline Chromo& get_locus (const int &x)	ｘ番目の個体の遺伝子
	double get_locus_value (const int &x)	ｘ番目の個体の設計変数の値
	int get_plan_num()	Plan_num の値を出力
	double& get_c(int i)	i 番目のスケーリング評価値の出力
	double& get_e(int i)	i 番目の評価値の出力
	Double& get_f()	適合度値の出力
	Inline void set_size(set_t n)	遺伝子を遺伝子長ｎで初期化
	Inline int size()	遺伝子の総数

	Inline static int get_chromo_len()	遺伝子長の値を出力
	Vector<Chromo>::iterator erase (vector<Chromo>::iterator n)	n番目の遺伝子を削除
	Vector<Chromo>::iterator begin()	vector演算子,beginの定義
	Inline static void set_obj_num(const int& x)	Obj_numの値を入力
	Static void set_parameter (const int& x,const int& y, const int& z)	各種パラメータを入力 x：目的関数の数 y：設計変数の数 z：遺伝子
	Static void set_func_num (const int& func_num)	Func_numの値を入力
	Inline int get_obj_num()	Obj_numの値を入力
	Inline int GetRnaLength()	遺伝子のバイト数を出力（Rna_sizeを出力）
	Inline int get_size()	総遺伝子数の出力
	int GetRnaLength_set()	遺伝子のバイト数を計算（Rna_sizeを設定）
	Gene operator=(const Gene&)	演算子「＝」の定義
	Chromo& operator() (const int& n)	演算子「()」の定義
	Chromo& operator[] (const int& n)	演算子「[]」の定義
	Virtual bool operator ==(const Gene&)	演算子「＝＝」の定義
	Virtual const u8* transcription(u8 *buffer)	クラスGeneのデータを1つの変数にまとめる（パック）
	Virtual void reverse_transcription (const u8 *data)	関数transcriptionによってまとめられたデータをクラスGeneのデータに分解する（アンパック）

	Virtual void reverse_transcription2 (const u8 *data)	関数 transcription によってまとめられたデータをクラス Gene のデータに分解する（アンパック）
場所	gene.h	

染色体の表現（Chromo クラス）

クラス名	Chromo	
内容	染色体を表現	
メンバ変数	u8 locus_r	遺伝子座
メンバ関数	Chromo operator =(const Chromo &f)	演算子「＝」の定義
	Chromo operator=(const u8 &f)	演算子「＝」の定義
	Chromo operator=(const int &f)	演算子「＝」の定義
	operator u8()	型変換
	operator int()	型変換
	bool operator ==(const Chromo &f)	演算子「＝＝」の定義
	bool operator==(const u8 &f)	演算子「＝＝」の定義
	U8 operator^(const Chromo&f)	演算子「＾」の定義
	U8 operator^(u8 &f)	演算子「＾」の定義
	void f_out(FILE *fp)	ファイルの出力
場所	oneisland.h	

評価値，適合度値の表現（Obj クラス）

クラス名	Obj	
内容	を表現	
メンバ変数	static int obj_num	目的関数の数

	double fitness_value	適合度値
	double *eval_value	評価値
	double *const_eval	スケーリング後の評価値
メンバ関数	Obj()	コンストラクタ
	Obj(int)	コンストラクタ
	Obj(const Obj&)	コンストラクタ
	~Obj()	デストラクタ
	void copy(const Obj&)	クラス Obj のコピー
	double& operator[](int i)	演算子「[]」の定義
	double& get_c_eval(int i)	i 番目のスケーリング後の評価値を出力
	double& get_fit()	適合度の値を出力
	u8 operator <(const Obj &dum)	演算子「<」の定義
	u8 operator >(const Obj &dum)	演算子「>」の定義
	u8 operator <=(const Obj &dum)	演算子「<=」の定義
	u8 operator >=(const Obj &dum)	演算子「>=」の定義
	void Obj_printf()	評価値の値を出力
	void Obj_printf_curent()	スケーリング後の評価値の値を出力
	Obj operator=(const Obj&)	演算子「=」の定義
	inline static void set_obj_num(int x)	obj_num の値を入力
	inline static int get_obj_num()	obj_num の値を出力
場所	gene.h	

5.3 各クラスの主要な関数の説明

本節では，クラスごとに主要な関数の説明を行う．本プログラムでは，4 つの手法に対応するために様々な関数が用意されているが，そのうち，GA 操作に密接に関わる主要な関数は

ある程度限られている．本節では，GA 操作に直接的に関わる One，Cross,Mut,Select，Numerical_func の 5 つのクラスにおける主要な関数について解説を行う．

各関数の説明 (クラス One)

One クラスにおける重要な関数は，次の 2 つの関数である．

initGA()　：初期化に関する関数

cycle()　：探索の最も外側のループ

そこで，以下では上記の 2 つの関数を中心に関わる関数の階層関係を示しながら主要な関数について解説を行う．

関数の呼び出し関係

```
void One::initGA()
    └ void One::init_base()
          ├ void One::o_file()
          │     └ void One::set_parameter_default()
          └ void One::set_parameter()
```

関数名	One::initGA()	
内容	GA を実行するための環境設定を行う，Init_base(), pop.set_pop_num(), pop.init_ap_gene()を呼ぶ.	
引数	なし	
戻り値	なし	
場所	oeisland.cpp	

関数名	One::Init_base()	
内容	ファイルなどの設定を行う o_file(),pop.set_obj_num()を呼ぶ.	
引数	なし	
戻り値	なし	
場所	oeisland.cpp	

関数名	One::o_file()

内容	set_parameter_default()を呼び，全てのパラメータにデフォルト値を設定する．その後，ファイル"mof_ga.txt"を開き，指定されているパラメータを読み込みパラメータ値の上書きを行う．	
引数	なし	
戻り値	なし	
場所	oneisland.cpp	

関数名	**One::set_parameter_default()**	
内容	Default パラメータを変数に代入する．	
引数	なし	
戻り値	なし	
場所	oneisland.cpp	

関数名	**One::set_parameter()**	
内容	選択，交叉，突然変異などのパラメータを代入する．	
引数	なし	
戻り値	なし	
場所	oeisland.cpp	

```
void One::cycle()
    ├─ void One::all_open_w_file()
    ├─ void One::initialize()
    │       ├─ init_pop(Pop &);
    │       └─ file_init_pop(Pop &dum_pop, ifstream& fin);
    ├─ void One::fuction()
    ├─ void One::f_output()
    │       └─ void One::f_output(Pop& dum_pop)
    ├─ void One::next_1_gene()
    │       ├─ void One::cross_over()
    │       ├─ void One::mutation()
    │       ├─ void One::function()
    │       ├─ void One::selection()
    │       ├─ void One::f_output()
    │       │       └─void One::f_output(Pop& dum_pop)
    │       └─ void One::terminal_check()
    ├─ void One::finish_write()
    └─ void One::all_open_file_close()
```

関数名	One::cycle()	
内容	GA を 1 試行実行する.	
引数	なし	
戻り値	なし	
場所	oeisland.cpp	

関数名	One::all_open_w_file()	
内容	結果を出力するためのファイルを開く.	
引数	なし	
戻り値	なし	
場所	oeisland.cpp	

関数名	One::initialize()	
内容	変数の初期化，および個体数を指定後，個体群を生成のための init_pop()を呼ぶ.	
引数	なし	
戻り値	なし	

場所	oeisland.cpp

関数名	One::init_pop()	
内容	各個体を生成する Func.pop_create()を呼ぶ.	
引数	なし	
戻り値	なし	
場所	oeisland.cpp	

関数名	One::file_init_pop()	
内容	初期個体設定ファイルを用いた場合の母集団生成.	
引数	Pop &dum_pop	ifstream& fin
戻り値	なし	
場所	oeisland.cpp	

関数名	One::function()	
内容	各個体の評価を行う func.sunc()を呼ぶ.	
引数	なし	
戻り値	なし	
場所	oeisland.cpp	

関数名	One::next_1_gene()	
内容	GA における 1 世代の計算を行う．各 GA オペレータを実行する.	
引数	なし	
戻り値	なし	
場所	oneisland.cpp	

関数名	One::cross_over()	
内容	交叉を行う cross.cross_over()を呼ぶ.	
引数	なし	
戻り値	なし	
場所	oneisland.cpp	

関数名	One::mutation()	
内容	突然変異を行う mut.mut()を呼ぶ.	
引数	なし	

戻り値	なし	
場所	oneisland.cpp	

関数名	One::function()	
内容	各個体の評価を行う func.sunc()を呼ぶ.	
引数	なし	
戻り値	なし	
場所	oneisland.cpp	

関数名	One::selection()	
内容	選択を行う select.selection()を呼ぶ.	
引数	なし	
戻り値	なし	
場所	oneisland.cpp	

関数名	One::f_output()	
内容	得られた解のファイルへの出力を行う.	
引数	なし	
戻り値	なし	
場所	oneisland.cpp	

関数名	One::f_output	
内容	引数として与えられた母集団 sum_pop に対してファイルへの出力を行う	
引数	Pop& sum_pop	母集団クラス
戻り値	なし	
場所	oneisland.cpp	

関数名	One::terminal_check()	
内容	終了条件を満たしているかを判定する.	
引数	なし	
戻り値	なし	
場所	oneisland.cpp	

関数名	One::finish_write()	
内容	全算時間，試行辺りの計算時間など探索が最終的に終了した段階でのファイル出力を行う	
引数	なし	
戻り値	なし	
場所	oneisland.cpp	

関数名	One::all_open_file_close()	
内容	現在開いているファイルをすべて閉じる.	
引数	なし	
戻り値	なし	
場所	oneisland.cpp	

各関数の説明（CrossOver）

Cross 関数における最も重要な関数は，**cross_over()**関数である．そこで，以下では**cross_over()**関数を中心にこの関数に関わる下位層に位置する関数について説明する．

```
void Cross::cross_over()
    └─ void Cross::cross_over(Gene **,int &)
            ├─ void Cross::random_shuffle()
            └─ void Cross::cross_method_exe()
                    ├─void Cross::One_point()
                    └─void Cross::Two_point()
```

関数名	Cross::cross_over()	
内容	交叉を行う関数 Cross::cross_over()を呼ぶ.	
引数	Pop& a	母集団クラス
戻り値	なし	
場所	cross_over.cpp	

関数名	Cross_cross_over()	
内容	交叉を行うための各関数を呼ぶ.	
引数	Gene **ap_gene int &num	遺伝子の 2 重ポインタ 遺伝子の個数

戻り値	なし	
場所	cross_over.cpp	

関数名	Cross::ramdom_shuffle	
内容	与えられた個体をランダムに並び替える	
引数	Gene **ap_gene const int &tmp_number	遺伝子の 2 重ポインタ 遺伝子の個数
戻り値	なし	
場所	cross_over.cpp	

関数名	Cross::cross_method_exe()	
内容	実際に交叉を行う．cross_method が 1 であれば 1 点交叉，それ以外であれば 2 点交叉を行う．	
引数	Gene **ap_gene const int &num	遺伝子の 2 重ポインタ 遺伝子の個数
戻り値	なし	
場所	cross_over.cpp	

関数名	Cross::One_point()	
内容	一点交叉	
引数	なし	
戻り値	なし	
場所	cross_over.cpp	

関数名	Cross::Two_point()	
内容	2 点交叉	
引数	なし	
戻り値	なし	
場所	cross_over.cpp	

各関数の説明（Mutation）

Mutation クラスは，GA 操作における突然変異を担っているクラスである．Mutation に

おける中心となっている関数は mut()である．そこで，以下では mut()関数を中心に関連するメンバ関数について解説する．
＃尚，本プログラムではビット反転にのみ対応している．

```
void Mutation::mut()
    └ void Mutation::mut(Gene **ap_gene,const int &num)
        └ void Mutation::mut_exe()
            └void Mutation::bit_reverse()
```

関数名	Mutation::mut()	
内容	突然変異を行う関数を呼ぶ．	
引数	Pop &a	母集団クラス
戻り値	なし	
場所	mutation.cpp	

関数名	Mutation::mut(Gene **ap_gene,const int &num)	
内容	突然変異率から突然変異をするかどうかを決定し，する場合には Mutation::mutation_exe()を実行する．	
引数	Gene **ap_gene const int &num	遺伝子の 2 重ポインタ 遺伝子の個数
戻り値	なし	
場所	mutation.cpp	

関数名	Mutation::mutation_exe()	
内容	突然変異を実行する．	
引数	Gene &gene	母集団のクラス
戻り値	なし	
場所	mutation.cpp	

関数名	Mutation::bit_reverse()	
内容	ビット反転を行う．	
引数	Gene &a	遺伝子クラス
戻り値	なし	

場所	mutation.cpp

各関数の説明（Select）

Select クラスにおける最も重要な関数は，selection()関数である．Selection()関数は，GA 操作における選択に相当する関数であり，そこから手法パラメータ select_method により幾つかの選択手法へと分岐している． 以下，selection()関数を中心にこの関数の下位関数について説明する．

```
void Select::selection(Pop& a)
    └ void Selection::selection(Gene **ap_gene,int &num)
          ├ void Select::ranking_con(Pop& a)
          │      └void Select::ranking_con(Gene **ap_gene,const int &num)
          ├ void Select::roulet_select(Gene **ap_gene, const int &num)
          ├ void Select::ranking_square(Gene **ap_gene, int &num)
          ├ void Select::obj_sharing()
          │      └void Select::obj_sharing
          │        (Gene **ap_gene,int &dum_pop_num,const int &dum)
          └ void Select::bi_select_alpha()
                 └void Select::bi_select_alpha
                   (Gene **ap_gene,int &dum_pop_num,const int &dum_dum)
```

関数名	Select::selection(Pop& a)	
内容	選択を行う関数を呼ぶ	
引数	Pop& a	母集団クラス
戻り値	なし	
場所	select.cpp	

関数名	Select::selection(Gene **ap_gene,int &num)	
内容	選択を行う関数を呼ぶ	
引数	Gene **ap_gene int &num	遺伝子の２重ポインタ 遺伝子の個数
戻り値	なし	
場所	select.cpp	

関数名	Select::ranking_con(Pop& a)	
内容	適合度割り付けの関数を呼び出す	
引数	Pop& a	母集団クラス
戻り値	なし	
場所	select.cpp	

関数名	Select::ranking_con(Gene **ap_gene,const int &num)	
内容	ランク付けされた個体にそのランクをもとに適合度を割り付ける	
引数	Gene **ap_gene const int &num	遺伝子の２重ポインタ 遺伝子の個数
戻り値	なし	
場所	select.cpp	

関数名	Select::roulet_select_normal(Gene **ap_gene, const int &num)	
内容	ルーレット選択を行う	
引数	Gene **ap_gene const int &num	遺伝子の２重ポインタ 遺伝子の個数
戻り値	なし	
場所	select.cpp	

関数名	Select:: obj_sharing(Pop& a)	
内容	目的関数空間シェアリングを行う関数を呼び出す	
引数	Pop& a	母集団クラス
戻り値	なし	
場所	select.cpp	

関数名	Select:: obj_sharing(Gene **ap_gene, int &dum_pop_num,const int &dum)	
内容	各個体の目的関数における値を用いてシェアリングを行う	
引数	Gene **ap_gene	遺伝子の２重ポインタ

	int &dum_pop_num const int &dum	遺伝子の個数 シェアリングパラメータ
戻り値	なし	
場所	select.cpp	

関数名	Select::ranking_square(Gene **ap_gene, int &num)	
内容	ランク付けされた各個体のランクの値を各個体のランクの２乗値に変更する	
引数	Gene **ap_gene int &num	遺伝子の２重ポインタ 遺伝子の個数
戻り値	なし	
場所	select.cpp	

関数名	Select:: bi_select_alpha(Pop &pop)	
内容	２個体トーナメント選択を行う関数を呼び出す	
引数	Pop &pop	母集団クラス
戻り値	なし	
場所	select.cpp	

関数名	Select:: bi_select_alpha(Gene **ap_gene, int &dum_pop_num,const int &dum_dum)	
内容		
引数	Gene **ap_gene int &dum_pop_num const int &dum_dum	遺伝子の２重ポインタ 遺伝子の個体数 選択する個体数
戻り値	なし	
場所	select.cpp	

各関数の説明（Numerical_func）

Numerical_func クラスは，Func クラスの派生クラスであり連続テスト関数を評価関数

とした場合の，GA 操作における評価部分を行っている．このクラスの最も主要な部分は，func()関数であり，この部分が実際の評価を担当している．以下，func()を中心にその下位関数について説明する．

```
void Numerical_func::func(Pop& a)
    └ void N umerical_func::func(Gene **ap_gene, const int &dum_pop)
          └void Numerical_func::func(Gene &ap_gene)
                ├ void Numerical::func1(Gene &ap_gene)
                ├ void Numerical::func2(Gene &ap_gene)
                ├ void Numerical::func3(Gene &ap_gene)
                ├ void Numerical::func4(Gene &ap_gene)
                ├ void Numerical::func5(Gene &ap_gene)
                ├ void Numerical::func6(Gene &ap_gene)
                ├ void Numerical::func7(Gene &ap_gene)
                ├ void Numerical::func8(Gene &ap_gene)
                ├ void Numerical::func9(Gene &ap_gene)
                ├ void Numerical::func10(Gene &ap_gene)
                ├ void Numerical::func11(Gene &ap_gene)
                ├ void Numerical::func12(Gene &ap_gene)
                └ void Numerical::func13(Gene &ap_gene)
```

関数名	**void Numerical_func::func(Pop& a)**	
内容	目的関数を呼び出す関数を呼び出す	
引数	Pop& a	母集団クラス
戻り値	なし	
場所	func_numerial.cpp	

関数名	**void Numerical_func::func(Gene **ap_gene, const int &dum_pop)**	
内容		
引数	Gene **ap_gene const int &dum_pop	遺伝子の２重ポインタ
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical_func::func(Gene &ap_gene)	
内容	目的関数を呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func1(Gene &ap_gene)	
内容	２．４．３節のテスト関数，玉置の例題２を呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func2(Gene &ap_gene)	
内容	２．４．３節のテスト関数，玉置の例題３を呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func3(Gene &ap_gene)	
内容	２．４．３節のテスト関数，玉置の例題４を呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func4(Gene &ap_gene)	
内容	２．４．３節のテスト関数，veldhuizen の３目的関数を呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func5(Gene &ap_gene)	
内容	２．４．３節のテスト関数，玉置の例題１を呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func6(Gene &ap_gene)	
内容	２．４．３節のテスト関数，Deb の考案した多峰性のある問題を呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func7(Gene &ap_gene)	
内容	２．４．３節のテスト関数，Deb の考案した偏重パレートフロントを呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func8(Gene &ap_gene)	
内容	２．４．３節のテスト関数，凹面多目的問題を呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func9(Gene &ap_gene)
内容	２．４．３節のテスト関数，Deb の考案した不連続多目的問題を呼び出す

引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func10(Gene &ap_gene)	
内容	２．４．３節のテスト関数，ZDT4 を呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func11(Gene &ap_gene)	
内容	２．４．３節のテスト関数，ZDT6 を呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func12(Gene &ap_gene)	
内容	２．４．３節のテスト関数，SPH-mを呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

関数名	void Numerical::func13(Gene &ap_gene)	
内容	２．４．３節のテスト関数，KUR を呼び出す	
引数	Gene &ap_gene	遺伝子の情報
戻り値	なし	
場所	func_numerial.cpp	

4.4 NCGA におけるアルゴリズムの説明

本節では,

5. 本プログラムを用いた 2 次利用について

本プログラムを用いて 2 次利用を行う場合，その目的によって変更を加える必要のあるクラスが異なる．以下，大まかな場合(Case)と変更クラスの関係について示す．

[Case 1 : 評価部分の変更について]

評価部分を変更する場合，Func クラスを継承する評価の派生クラスを作成する必要がある．連続関数をテスト問題とする場合には，Numerical_func クラスを参考に新たな Func クラスの派生クラスを作成するか，Numerical_func クラスそのものを書き換えることで対応することができる．

ただし，新たな Func クラスの派生クラスを作成した場合には Oneisland.h ファイル中にあるメンバ変数 Numerical_func クラスの部分を新たな派生クラス名に変更する必要がある．同時に，新たな派生クラスのヘッダファイルの読み込み(include)も行う必要がある．また，新たなファイルを作成した場合にはそれに応じて makefile の書き換えも必要となる．

[Case2: 遺伝子の表現を変更したい]

遺伝子表現(コーディング)を変更する場合, Gene クラスおよび Chromo クラスの変更を行う必要がある．また，遺伝子の表現を変更する場合，必然的に Func クラスの変更(新たな派生クラスの作成)が必要となる．これは, 例え同じ問題を用いていてもビットコーディングの場合と実数値コーディングの場合では, 新たに Func 部分を書き直さなければならないことからも分かる．

本プログラムでは， Gene クラスと Chromo クラスは同じ gene.{cpp，h}ファイルに書かれている． 以下，本プログラムにおけるビットコーディングを実数値コーディングへ変更する場合の方法について説明する．

1. Chromo クラス の書き換え

   ビットコーディングの場合,「u8 (unsigned char)」で定義されているメンバ変数部分を「double 」型へと変更する．それに伴い，以下の関数を削除する．

   operator int() {return (int)locus_r;}

   operator u8() {return locus_r;}

   u8 operator^(const Chromo&);

   また，以下の関数に変更を加える必要がある．

   f_out(FILE *fp) {fprintf(fp,"%d ",locus_r); から fprintf(fp,"%f ",locus_r);へと変更}

   さらに，以下の関数を新たに加える必要がある．

operator double() {return locus_r;}

2. Gene クラスの書き換え

   Gene クラスではヘッダファイル自体は変更する必要は無い．しかし，以下の関数について変更を加える必要がある．

   double Gene::get_locus_value(const int& point) const

   double Gene::get_variable_value(const int& start,const int& end) const

   int Gene::GetRnaLength_set()

   const u8* Gene::transcription(u8 *buffer)

   void Gene::reverse_transcription(const u8 *data)

   void Gene::reverse_transcription2(const u8 *data)

   また，設計変数の値と染色体の値に違いがなくなるため **f_chromosome(FILE *fp)** と **f_show_plan_value(FILE *fp)**は同一の中身となる．

3. Func クラス(Numerical_func)の書き換え

   ここでは, 新たな Func クラスの派生クラスを作成するのでは無く, Numerical_func クラスを書き換えることにより対応する方法について説明する．

   Numerical_func クラスにおいて変更が必要となるメンバ関数を以下に示す．

   void Numerical_func::pop_create(Gene **ap_gene,const int& num)

[Case ３: 交叉・突然変異を変更したい]

交叉方法，突然変異方法を変更する方法として，大きく以下の 2 つがある．

A) Cross クラス，Mutation クラスの派生クラスを作成する．

B) Cross クラス，Mutation クラスそのものに変更を加える．

ビットコーディングではない別のコーディング方法を用いた場合には，新たな派生クラスを作成したほうが良いと思われる．この場合，Cross および Mutation の派生クラスとして(コンストラクタ，デコンストラクタ以外に)最低限必要となるメンバ関数は,

virtual void cross_over(Pop& a);

および,

virtual void mut(Pop &);

である．

逆にビットコーディングを用いた交叉手法，突然変異手法の変更であれば B)の方法を用いた方が良いと思われる．この場合には，Cross クラス，Mutation クラスの変更を行うべきメンバ関数は，それぞれ

virtual void cross_method_exe(Gene **,const int &);

virtual void mutation_exe(Gene &gene);

である．また上記に変更に伴う新たな交叉方法，突然変異方法に関するメンバ関数を新たに加える必要がある．

参考文献

[1] Kalyanmoy Deb, "Multi-Objective Optimization using Evolutionary Algorithms", Chichester, UK: Wiley, 2001.

[2] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization", Proceedings of the 5th international coference on genetic algorithms, pp. 416—423, 1993.

[3] E. Zitzler, M. Laumanns and L. Thiele, ” SPEA2: Improving the Performance of the Strength Pareto Evolutionary Algorithm”, Technical Report 103, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich, 2001.

[4] K. Deb, S. Agarwal, A. Pratap and T. Meyarivan, " A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II", KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.

[5] K. Deb and T. Goel, "Controlled Elitist Non-dominated Sorting Genetic Algorithms for Better Convergence", First International Conference on Evolutionary Multi-Criterion Optimization, pp. 67-81, 2001.

[6] S. Watanabe, T. Hiroyasu and M. Miki, "Evolutionary multi-criterion optimization for mobile telecommunication networks optimization", EUROGEN 2001 - Evolutionary Methods for Design Optimisation and Control with Applications to Industrial Problems, 2001.

[7] T. Hiroyasu, M. Miki and S. Watanabe, "The New Model of Parallel Genetic Algorithm in Multi-Objective Optimization Problems -Divided Range Multi-Objective Genetic Algorithms",IEEE Proceedings of the 2000 Congress on Evolutionary Computation,pp. 333-340, 2000.

[8] 玉置, 森, 荒木, "遺伝アルゴリズムを用いたパレート最適解集合の生成法", 計測自動制御学会論文集, vol. 31, number. 8, pp. 1185—1192, 1995.

[9] Kalyanmoy Deb and T. Meyarivan, "Constrained Test Problems for Multi-Objective Evolutionary Optimization", KanGAL report 200005, Indian Institute of Technology, Kanpur, India, 2000.

[10] Eckart Zitzler, Kalyanmoy Deb and Lothar Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results",

Evolutionary Computation, volume. 8, number 2, pp. 173-195, 2000.

[11] D. A. V. Veldhuizen and G. B. Lamont, "Multiobjective Evolutionary Algorithm Test Suites",Proceedings of the 1999 ACM Symposium on Applied Computing, pp. 351—357, 1999.

[12] Frank Lirsawe, "A Variant of Evolution Strategies for Vector Optimization", PPSN I, volume 496 of Lecture Notes in Computer Science, pp. 193-197, 1991.

Copyright

Multi Objective Optimizaion Team of Intelligent Systems Design Laboratory, Doshisha University.