1　はじめに

本研究では，新しい品質多様性アルゴリズムである ABC MAP-Elites の提案と，その有効性を検証するための実験を行った．ここでは、研究で使用した計算機環境からプログラムの実行環境の構築、およびソースコードに関する情報をまとめる。まず、2 節では、実験で使用した計算機環境について述べる。次に、3 節では、本研究で作成したソースコードについて説明する。

2　セットアップ

2.1　計算機環境

実験で用いる計算機環境を表1に示す．OSはUbuntu 22.04 LTS，CPUはIntel(R) Core(TM) i7-13700F，メモリは32GiBである．本実験のプログラムは，すべてDockerのコンテナ上で実行する．　使用するプログラミング言語はJulia 1.10.5である．プログラムで用いた

表 1: 計算機環境

OS		Ubuntu jammy 22.04 x86_64
Kernel		Linux 5.15.0-124-generic
CPU	型名	13th Gen Intel(R) Core(TM) i7-13700F @ 5.20 GHz
	コア数	24
メモリ		32.0 GiB
使用ソフトウェア		Julia 1.10.5 Docker 27.4.1

Juliaのパッケージとバージョンを表2に示す．DelaunayTriangulation[1]はCVTの計算に使用する．Distributions[2, 3]は正規分布の生成に使用する．StableRNGs[4]は乱数生成に使用する．FileIO[5]とJLD2[6]はファイル入出力に使用し，CairoMakie[7]はグラフやボロノイ図のプロットに使用する．

表 2: 使用したパッケージ

パッケージ	バージョン	詳細
DelaunayTriangulation	1.6.3	CVT の計算
StableRNGs	1.0.2	乱数生成
Distributions	0.25.116	正規分布の生成
LinearAlgebra	-	線形代数の標準ライブラリ
FileIO	1.16.6	ファイル入出力
JLD2	0.5.10	ファイル入出力
CairoMakie	0.12.18	図のプロット

2.2　初期設定

プログラムの実行方法について説明する．まず，ソースコード 1 の Makefile を用いて，Docker のインストールを行う．コマンドは `$ make container` である．

ソースコード 1: Makefile

```
.PHONY: all install-docker install-julia install-python install-jinja2
all: install-docker install-julia install-python install-jinja2
container: install-docker install-python install-jinja2
local: install-julia
install-docker:
  sudo apt-get update
  sudo apt-get install -y \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
  sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
  sudo apt-get update
  sudo apt-get install -y docker-ce
  sudo gpasswd -a $USER docker
  newgrp docker
install-julia:
  curl -fsSL https://install.julialang.org | sh
  . ~/.bashrc
  julia pkginstall.jl
install-python:
  sudo apt -y install python3-pip
install-jinja2:
  pip install jinja2
```

Docker のインストールが完了したら，`$ docker -v` を実行してバージョンが表示されることを確認する．続いて，ソースコード 2 の `Dockerfile` と ソースコード 3 の `docker-compose-run.yaml` を用いて，Julia のコンテナをデプロイする．

ソースコード 2: Dockerfile

```
FROM julia:1.10.5

ARG lang="C"
ARG dir="src"
ENV DEBIAN_FRONTEND noninter active
ENV LANG ${lang}
ENV TZ Asia/Tokyo

WORKDIR /root
COPY pkginstall.jl /root/pkginstall.jl

RUN apt -y update && apt -y upgrade &&\
  julia pkginstall.jl run &&\
  rm -rf pkginstall.jl

WORKDIR /root/${dir}
COPY ./${dir}/*.jl /root/${dir}
```

ソースコード 3: docker-compose-run.yaml

```
services:
  julia-run-abc-sphere-3:
    container_name: "julia-run-abc-sphere-3"
    tty: true
    build:
      context: .
      dockerfile: Dockerfile
    working_dir: /root/src
    volumes:
      - ./src:/root/src
    networks:
      - default
    environment:
      - FUNCTION=sphere
      - METHOD=abc
      - D=10
    deploy:
    resources:
      limits:
        cpus: 1.0
```

```
21            memory: 8G
22        command: ["julia", "main.jl", "$$D", "$$METHOD", "cvt", "$$FUNCTION", "3"]
23    networks:
24      default:
25        driver: bridge
```

デプロイを行うコマンドは $ docker compose -f docker-compose-run.yaml up である．Julia のコンテナが起動したら，$ docker ps を実行してコンテナが起動していることを確認する．出力結果は ./src/result/$METHOD/$FUNCTION に保存される．

3　ソースコード

本実験で使用したソースコードについて解説する．なお、すべてのソースコード全体については、付録 C のソースコード 28 からソースコード 39 までを参照されたい．

3.1　基本ファイル

ここでは、主に基本的な機能に関するソースコードについて説明する．

main.py

ソースコード 4 に示すのは、本研究で使用したメイン関数である．メイン関数では，探索アルゴリズムの実行から結果の保存までを行う．ソースコード 4 の 7 行目から 15 行目では，実行時間の計測と MAP-Elites アルゴリズムの実行を行っている．`main.jl` 全体はソースコード 28 に記載している．

ソースコード 4: main 関数

```
1   function main()
2       # Make result directory and log file
3       MakeFiles()
4
5       CheckParameters()  # パラメータをチェック
6
7       #------ MAP ELITES ALGORITHM --------------------------------#
8
9       begin_time = time()                    # 開始時間を記録
10
11      popn, arch, iter_time = map_elites()  # MAP-アルゴリズムを実行Elites
12
13      finish_time = time()                   # 終了時間を記録
14
15      #------ MAP ELITES ALGORITHM --------------------------------#
16
```

```
17          elapsed_time = finish_time - begin_time   # 経過時間を計算
18
19          println("End␣of␣Iteration.\n")
20          println("Time␣of␣iteration:␣", iter_time, "␣[sec]")      # 反復の時間を出力
21          println("Time:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣", elapsed_time, "␣[sec]")  # 総経過時間を出力
22
23          # Save result
24          SaveResult(arch, iter_time, elapsed_time)
25      end
```

config.jl

ソースコード 30 に示すのは、本研究で使用したパラメータ設定ファイルである．パラメータ設定ファイルでは，ソースコード全体のすべてのパラメータを設定および管理している．詳細は表 3 から表 9 までを参照されたい．

表 3: 基本のパラメータ

パラメータ名	説明	デフォルト値	備考
D	次元数	ARGS[1]	ARGS[1] が"test"の場合は 2 に固定
N	集団サイズ	64	
BD	行動次元数	2	変更不可
CONV_FLAG	収束フラグ	false	true の場合、収束確認モード
ε (EPS)	収束判定の閾値	1e−6	
FIT_NOISE	フィットネスにノイズを追加するか	true	
r_{noise}	ノイズ率	0.01	
MAXTIME	最大時間ステップ数	条件により変化	CONV_FLAG や OBJ_F に依存

表 4: 行動空間の生成に関するパラメータ

パラメータ名	説明	デフォルト値	備考
GRID_SIZE	グリッドマップのグリッドサイズ	158	MAP_METHOD == grid 時に使用
$k_{\max}$	CVT 方式の最大クラスタ数	25000	MAP_METHOD == cvt 時に使用

表 5: CVT のパラメータ

パラメータ名	説明	デフォルト値	備考
cvt_vorn_data_update_limit	Voronoi データの更新制限	3	
CVT_MAX_ITER	CVT の最大反復回数	100	

表 6: MAP-Elites アルゴリズムのパラメータ

パラメータ名	説明	デフォルト値	備考
MUTANT_R	突然変異率	0.90	

表 7: DME のパラメータ

パラメータ名	説明	デフォルト値	備考
CR	交叉確率。	条件により変化	OBJ_F に依存
F	スケーリングファクタ。	条件により変化	OBJ_F に依存

表 8: DE のパラメータ設定

関数名	CR	F
Sphere	0.2	0.4
Rosenbrock	0.7	0.8
Rastrigin	0.5	0.6

表 9: ABCME のパラメータ

パラメータ名	説明	デフォルト値	備考
FOOD_SOURCE	ABC の食料源（探索限界トライアル数）	N	
TC_LIMIT	ABC の探索限界トライアル数	$D \cdot \lfloor \frac{k_{\max}}{10 \cdot \texttt{FOOD\_SOURCE}} \rfloor$	

benchmark.jl

ソースコード 29 に示すのは、本研究で使用したベンチマーク関数である．ベンチマーク関数は，探索アルゴリズムの性能を評価するために使用される．本研究では，Sphere, Rosenbrock，Rastrigin の 3 つのベンチマーク関数を使用している．表 10 にベンチマーク関数の数式を示す.

表 10: ベンチマーク関数とその数式

関数名	式	x_i の定義域	最小値
Sphere	$f(\boldsymbol{x}) = \sum_{i=1}^{D} x_i^2$	$[-5.12, 5.12]$	$f(0, \ldots, 0) = 0$
Rosenbrock	$f(\boldsymbol{x}) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right]$	$[-5.00, 5.00]$	$f(1, \ldots, 1) = 0$
Rastrigin	$f(\boldsymbol{x}) = 10D + \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i)\right]$	$[-5.12, 5.12]$	$f(0, \ldots, 0) = 0$

struct.jl

ソースコード 5, 6, 7 に示すのは，本研究で使用した構造体の定義である．ソースコード 5 の `Individual` 構造体は，個体を表す構造体である．`genes` は，個体の遺伝子を表すベクトルである．`benchmark` は，個体のベンチマーク値を表すタプルであり，要素 1 はノイズありのベンチマーク値，要素 2 はノイズなしのベンチマーク値である．`behavior` は，個体の行動空間を表すベクトルである．

ソースコード 5: Individual 構造体

```
1    mutable struct Individual
2        # N dimension vector
3        genes::Vector{Float64}
4        # Benchmark value (1: with noise, 2: without noise)
5        benchmark::Tuple{Float64, Float64}
```

```
6         # Behavior space
7         behavior::Vector{Float64}
8     end
```

ソースコード 6 の Population 構造体は，個体群を表す構造体である．individuals は，個体の集合を表すベクトルであり，Individual 構造体のベクトルである．

ソースコード 6: Population 構造体

```
1     mutable struct Population
2         # Group of individuals | 個体群
3         individuals::Vector{Individual}
4     end
```

ソースコード 7 の Archive 構造体は，行動空間のアーカイブを表す構造体である．grid は，行動空間のグリッドマップを表す行列である．grid_update_counts は，グリッドの更新回数を表すベクトルである．individuals は，個体の集合を表す辞書であり，Individual 構造体の辞書である．

ソースコード 7: Archive 構造体

```
1     mutable struct Archive
2         # Grid map | グリッドマップ
3         grid::Matrix{Int64}
4         # Grid update counts | グリッド更新回数
5         grid_update_counts::Vector{Int64}
6         # Individuals | 個体
7         individuals::Dict{Int64, Individual}
8     end
```

logger.jl

ソースコード 32 に示すのは、本研究で使用したログファイルの作成関数である．ログファイルは，プログラムの実行状況を記録するために使用される．

savedata.jl

ソースコード 33 に示すのは，本研究で使用した測定結果を保存するプログラムである．測定結果は，アーカイブの行動空間，ベンチマーク値，最良解と準最適解の結果などを保存する．

cvt.jl

ソースコード 34 に示すのは，本研究で使用した CVT に関するプログラムである．CVT マッピング関数は，行動空間をグリッドにマッピングするために使用される．CVT マッピング関数は，`init_CVT` 関数と `cvt_mapping` 関数の 2 つの関数で構成される．

ソースコード 8 の `init_CVT` 関数は，CVT マッピングを初期化する関数である．まず，5 行目でランダムな点を生成し，7 行目で取得した個体の行動識別子を 9 行目の配列 `points` に追加する．さらに 11 行目で取得した境界点を同様に追加する．14 行目で `vorn` にボロノイ図を生成し，保存する．`cvt_vorn_data_update` をインクリメントして，ログを記録する．最後に，ボロノイ生成点 `vorn` を返す．

ソースコード 8: init_CVT 関数

```
1   function init_CVT(population::Population)
2       global vorn, cvt_vorn_data_update
3
4       # ランダムな点を生成
5       points   = [rand(RNG, BD) .* (UPP - LOW) .+ LOW for _ in 1:k_max - (N + 4)]
6       # 個体の行動識別子を取得
7       behavior = [population.individuals[i].behavior for i in 1:N]
8       # 行動識別子を点に追加
9       append!(points, behavior)
10      # 境界点を追加
11      append!(points, [[UPP, UPP], [UPP, LOW], [LOW, UPP], [LOW, LOW]])
12
13      # ボロノイ図の生成と保存
14      vorn = centroidal_smooth(voronoi(triangulate(points; rng = RNG), clip = false);
          maxiters = CVT_MAX_ITER, rng = RNG)
```

```
15          save("$(output)$(METHOD)/$(OBJ_F)/CVT-$(FILENAME)-$(cvt_vorn_data_update).jld2", "
              voronoi", vorn)
16
17          cvt_vorn_data_update += 1   # 更新カウンタをインクリメント
18          logger("INFO", "CVT␣is␣initialized")   # 初期化完了のログを記録
19
20          # ボロノイ生成点を返す
21          return DelaunayTriangulation.get_generators(vorn)::Dict{Int64, Tuple{Float64,
              Float64}}
22      end
```

続いて，ソースコード 9 の cvt_mapping 関数は，CVT マッピングを行う関数である．6 行目では distances に各生成点との距離を計算し，最も近い生成点のインデックスを取得する．13 行目の closest_centroid_index には distances が最小となる生成点のインデックスが格納される．16 行目から 23 行目では，アーカイブに生成点が存在する場合，個体の評価がアーカイブよりも良い場合は，新しい個体をアーカイブに追加し，更新カウンタをインクリメントする．24 行目から 30 行目では，アーカイブに生成点が存在しない場合，新しい個体をアーカイブに追加し，更新カウンタをインクリメントする．

ソースコード 9: cvt_mapping 関数

```
1   function cvt_mapping(population::Population, archive::Archive)
2       global vorn
3
4       for ind in population.individuals
5           # 各生成点との距離を計算
6           distances = [
7               norm(
8                   [ind.behavior[1] - centroid[1], ind.behavior[2] - centroid[2]],
9                   2
10              ) for centroid in values(DelaunayTriangulation.get_generators(vorn))
11          ]
12          # 最も近い生成点のインデックスを取得
13          closest_centroid_index = argmin(distances)
14
15          # アーカイブに生成点が存在する場合
16          if haskey(archive.individuals, closest_centroid_index)
```

```
17                  # 個体の評価がアーカイブよりも良い場合
18                  if ind.benchmark[fit_index] < archive.individuals[closest_centroid_index].
                      benchmark[fit_index]
19                      # 新しい個体をアーカイブに追加
20                      archive.individuals[closest_centroid_index] = Individual(deepcopy(ind.
                          genes), ind.benchmark, deepcopy(ind.behavior))
21                      # 更新カウンタをインクリメント
22                      archive.grid_update_counts[closest_centroid_index] += 1
23                  end
24              else  # アーカイブに生成点が存在しない場合
25                  # 新しい個体をアーカイブに追加
26                  archive.individuals[closest_centroid_index] = Individual(deepcopy(ind.genes
                      ), ind.benchmark, deepcopy(ind.behavior))
27                  # 更新カウンタをインクリメント
28                  archive.grid_update_counts[closest_centroid_index] += 1
29              end
30          end
31
32          return archive  # 更新されたアーカイブを返す
33      end
```

探索アルゴリズムと実行プログラム

ここでは、MAP-Elites アルゴリズムとそれに関連するソースコードについて説明する．

me.jl

ソースコード 37 に示すのは，本研究で使用した MAP-Elites アルゴリズムの実行プログラムである．このプログラムは，MAP-Elites, Differential MAP-Elites, ABC MAP-Elites の 3 つのアルゴリズムをそれぞれ実行する際に使用される．

ソースコード 10: devide_gene 関数

```
function devide_gene(gene::Vector{Float64})
    gene_len    = length(gene)        # 遺伝子の長さを取得
    segment_len = div(gene_len, BD)   # セグメントの長さを計算
    behavior    = Float64[]           # 行動ベクトルを初期化

    for i in 1:BD
        start_idx = (i - 1) * segment_len + 1              # 開始インデックスを計算
        end_idx   = i == BD ? gene_len : i * segment_len   # 終了インデックスを計算
        # 行動ベクトルに値を追加
        push!(behavior, BD*sum(gene[start_idx:end_idx])/Float64(gene_len))
    end

    return behavior   # 行動ベクトルを返す
end
```

ソースコード 10 は，行動識別子を生成する関数である．この関数は，遺伝子の長さを取得し，行動空間の次元数 BD に従いセグメントの長さを計算し，遺伝子をセグメントごとに分割する．行動識別子はセグメントの値を合計し，セグメントの長さで割ることで，行動空間の次元数 BD ごとの行動ベクトルを生成する．

ソースコード 11 は，個体を初期化する関数である．この関数は，ランダムな遺伝子を生成し，ノイズを加えて個体を生成する．最後に，生成した個体を構造体 `Individual` に格納して返す．例として，`best_solution` は，最良解を初期化するために使用される．

ソースコード 11: init_solution 関数

```
1  function init_solution()
2      gene = rand(RNG, D) .* (UPP - LOW) .+ LOW   # ランダムな遺伝子を生成
3      gene_noised = noise(gene)                   # ノイズを加える
4      # 個体を生成して返す
5      return Individual(deepcopy(gene_noised), (objective_function(gene_noised),
         objective_function(gene)), devide_gene(gene_noised))
6  end
7
8  best_solution = init_solution()  # 最良解を初期化
```

ソースコード 12 の evaluator 関数は，個体の評価を行う MAP-Elites を構成する重要な関数のひとつである．まず，4 行目から 6 行目で，目的関数の値を計算する．次に，7 行目から 8 行目で行動を評価し，9 行目から 12 行目で最良解より個体の評価が良い場合は最良解を更新する．最後に，評価された個体を返す．

ソースコード 12: evaluator 関数

```
1  function evaluator(individual::Individual)
2      global best_solution
3
4      # Objective function 目的関数の値を計算
5      gene_noised = noise(individual.genes)  # ノイズを加える
6      individual.benchmark = (objective_function(gene_noised), objective_function(
         individual.genes))
7      # Evaluate the behavior 行動を評価
8      individual.behavior = deepcopy(devide_gene(gene_noised))
9      # Update the best solution 最良解より個体の評価が良い場合、最良解を更新
10     if individual.benchmark[fit_index] <= best_solution.benchmark[fit_index]
11         best_solution = Individual(deepcopy(individual.genes), deepcopy(individual.
             benchmark), deepcopy(individual.behavior))
12     end
13
14     return individual  # 評価された個体を返す
15 end
```

ソースコード 13 の mapping 関数は，行動空間のマッピングを行う関数である．この関数は if 式による条件分岐を行い，MAP_METHOD が”cvt”の場合は cvt_mapping 関数 (ソースコード

34 の 55 行目) を実行し，それ以外の場合はエラーを出力して終了する．本来は MAP_METHOD が”grid”の場合も評価するが，本文では省略している．cvt_mapping 関数の説明はソースコード 9 の解説を参照されたい．

ソースコード 13: mapping 関数

```
    Mapping = if MAP_METHOD == "cvt"
        # CVT マッピング
        (population::Population, archive::Archive) -> cvt_mapping(population, archive)
    else
        error("Invalid␣MAP␣method")  # 無効なメソッドエラー -> 終了
        logger("ERROR", "Invalid␣MAP␣method")  # ログ出力
        exit(1)
    end
```

ソースコード 14 の mutate 関数は，個体の突然変異を行う関数である．この関数は，個体の遺伝子を突然変異率 MUTANT_R に従って突然変異させる．突然変異率 MUTANT_R よりも乱数が小さい場合は，遺伝子を突然変異させる．

ソースコード 14: mutate 関数

```
    mutate(individual::Individual) = Individual(
        # 突然変異
        [rand(RNG) < MUTANT_R ? rand(RNG) * (UPP - LOW) + LOW : gene for gene in individual
          .genes],
        # ベンチマーク値を初期化
        (0.0, 0.0),
        # 行動ベクトルを初期化
        zeros(Float64, BD)
    )
```

ソースコード 15 の select_random_elite 関数は，個体の選択を行う関数である．この関数は，アーカイブの個体群からランダムにエリート個体を選択する．

ソースコード 15: select_random_elite 関数

```
    select_random_elite(population::Population, archive::Archive) = archive.individuals[
      rand(RNG, keys(archive.individuals))]
```

ソースコード 16 の Reproduction 関数は，個体の探索を行う重要な関数のひとつである．先ほどの mapping 関数と同様に，if 式による条件分岐を行い，METHOD が”me”の場合は MAP-Elites，”de”の場合は Differential MAP-Elites，”abc”の場合は ABC MAP-Elites を実行する．2 行目から 8 行目の MAP-Elites の探索は，select_random_elite 関数でエリート個体を選択し，mutate 関数で突然変異を行い，evaluator 関数で個体を評価することで行われる．

ソースコード 16: Reproduction 関数

```
1   Reproduction = if METHOD == "me"
2       # MAP-Elites
3       (population::Population, archive::Archive) -> (
4           Population([
5               evaluator(mutate(select_random_elite(population, archive))) for _ in 1:N
6           ]),
7           archive
8       )
9   elseif METHOD == "de"
10      # Differential MAP-Elites
11      (population::Population, archive::Archive) -> DE(population, archive)
12  elseif METHOD == "abc"
13      # ABC MAP-Elites
14      (population::Population, archive::Archive) -> ABC(population, archive)
15  else
16      error("Invalid method")  # 無効なメソッドエラー -> 終了
17      logger("ERROR", "Invalid method")  # ログ出力
18      exit(1)
19  end
```

ソースコード 37 の map_elites 関数は，探索アルゴリズムを実行する関数である．ここでは，大まかに初期化と探索の 2 つのステップに分けて説明する．ソースコード 17 では，4 行目で Population 構造体を初期化し，7 行目で Archive 構造体を初期化している．6 行目では init_CVT 関数 (ソースコード 34 の 34 行目) を呼び出して，CVT マッピングを初期化している．

ソースコード 17: map_elites 関数の初期化

```
    logger("INFO", "Initialize")

    # Initialize the population | 個体群を初期化
    population::Population = Population([evaluator(init_solution()) for _ in 1:N])

    # Initialize the archive | CVT とアーカイブを初期化
    archive::Archive = if MAP_METHOD == "cvt"  # CVT マッピングの場合
        init_CVT(population)
        Archive(zeros(Int64, 0, 0), zeros(Int64, k_max), Dict{Int64, Individual}())
    else
        error("Invalid␣MAP␣method")  # 無効なメソッドエラー -> 終了
        logger("ERROR", "Invalid␣MAP␣method")  # ログ出力
        exit(1)
    end
```

続いて，ソースコード 18 では，探索アルゴリズムの探索ステップを説明する．探索は 9 行目から 15 行目の Evaluator，Mapping，Reproduction の 3 つのステップで構成される．Evaluator ステップでは，個体の評価を行い，Population 構造体に格納する．Mapping ステップでは，Population 構造体を Archive 構造体にマッピングする．Reproduction ステップでは，Population 構造体と Archive 構造体を用いて，個体の探索を行う．探索の反復回数は，MAXTIME で指定された回数まで行う．map_elites 関数は，更新された Population 構造体と Archive 構造体，および反復回数の経過時間を返す．

ソースコード 18: map_elites 関数の探索

```
    logger("INFO", "Start␣Iteration")
    begin_time = time()  # 初期化後の開始時間を取得

    for iter in 1:MAXTIME  # Iteration for MAXTIME
        # Print the generation
        println("Generation:␣", iter)

        # Evaluator
        population = Population([evaluator(ind) for ind in population.individuals])

        # Mapping
```

```
12          archive = Mapping(population, archive)
13
14          # Reproduction
15          population, archive = Reproduction(population, archive)
16
17          # Print the solutions
18          indPrint(ffn, ff)
19      end
20
21      finish_time = time()   # 終了時間を取得
22      logger("INFO", "Time out")
23
24      return population, archive, (finish_time - begin_time)   # 個体群、アーカイブ、経過時間を返す
```

de.jl

ソースコード 38 の de.jl に示すのは，本研究で使用した Differential MAP-Elites (DME) アルゴリズムの実行プログラムである．de.jl には，差分進化アルゴリズム (Differential Evolution, DE) が実装されており，ソースコード 16 の Reproduction 関数で呼び出される DE 関数が記述されている．DE 関数は，Reproduction フェーズにて探索を行う関数である．ソースコード 19 に示すのは，DE 関数の詳細である．まず，7 行目から 9 行目で，ランダムな異なるインデックスを生成し，ドナーベクトル r1，r2，r3 を選択する．12 行目で差分ベクトル v を計算し，14 行目でソースコード 20 の crossover 関数を用いて二項交叉を行い，トライアルベクトル u を計算する．16 行目と 17 行目で，ノイズを加えたトライアルベクトル u_noised を計算し，ベンチマーク値 b を計算する．20 行目から 33 行目では，ターゲットベクトルとドナーベクトル r1，r2，r3 の評価を比較し，アーカイブを更新する．34 行目から 36 行目では，ターゲットベクトルとトライアルベクトルの評価を比較し，個体を更新する．最後に，更新された Population 構造体と Archive 構造体を返す．

ソースコード 19: DE 関数

```
    function DE(population::Population, archive::Archive)
        I_p, I_a = population.individuals, archive.individuals  # 個体群とアーカイブの個体を取得
        r1, r2, r3 = zeros(Int, 3)                              # ランダムなインデックスを初期化
        b = Tuple{Float64, Float64}[]                           # タプルを初期化

        for i in 1:N
            while r1 == r2 || r1 == r3 || r2 == r3 || I_a[r1].genes == I_p[i].genes || I_a[
              r2].genes == I_p[i].genes || I_a[r3].genes == I_p[i].genes
                # ランダムな異なるインデックスを生成 -> ドナーベクトルを選択
                r1, r2, r3 = rand(RNG, keys(I_a), 3)
            end
            # 差分ベクトルを計算
            v = clamp.(I_a[r1].genes .+ F .* (I_a[r2].genes .- I_a[r3].genes), LOW, UPP)
            # 二項交叉を行い、トライアルベクトルを計算
            u = crossover(I_p[i].genes, v)

            u_noised = noise(u)  # ノイズを加える
            b = (objective_function(u_noised), objective_function(u))  # ベンチマークを計算

            # ターゲットベクトルとドナーベクトル r1 の評価を比較
            if b[fit_index] < I_a[r1].benchmark[fit_index]
                # アーカイブ r1 を更新
                archive.individuals[r1] = Individual(deepcopy(u), b, devide_gene(u))
            end
            # ターゲットベクトルとドナーベクトル r2 の評価を比較
            if b[fit_index] < I_a[r2].benchmark[fit_index]
                # アーカイブ r2 を更新
                archive.individuals[r2] = Individual(deepcopy(u), b, devide_gene(u))
            end
            # ターゲットベクトルとドナーベクトル r3 の評価を比較
            if b[fit_index] < I_a[r3].benchmark[fit_index]
                # アーカイブ r3 を更新
                archive.individuals[r3] = Individual(deepcopy(u), b, devide_gene(u))
            end
            # ターゲットベクトルとトライアルベクトルの評価を比較
            if b[fit_index] < I_p[i].benchmark[fit_index]
                # 個体を更新
                population.individuals[i] = Individual(deepcopy(u), b, devide_gene(u))
            end
        end

        return population, archive  # 更新された個体群とアーカイブを返す
```

```
42      end
```

ソースコード 20: crossover 関数

```
1       # 二項交叉
2       crossover(x::Vector{Float64}, v::Vector{Float64}) = [d == rand(RNG, 1:D) || rand(RNG) <
            CR ? v[d] : x[d] for d in 1:D]
```

abc.jl

ソースコード 39 の `abc.jl` に示すのは，本研究で使用した ABC MAP-Elites (ABCME) アルゴリズムの実行プログラムである．`abc.jl` には，人工蜂コロニーアルゴリズム (Artificial Bee Colony, ABC) が実装されており，ソースコード 16 の `Reproduction` 関数で呼び出される `ABC` 関数が記述されている．`ABC` 関数は，`Reproduction` フェーズにて探索を行う関数である．ソースコード 21 に示すのは，`ABC` 関数の詳細である．この関数は，ABC アルゴリズムの 3 つのフェーズである収穫蜂フェーズ，追従蜂フェーズ，偵察蜂フェーズを順に実行する．各フェーズの関数については，ソースコード 22 からソースコード 27 に示す．

ソースコード 21: ABC 関数

```
1       function ABC(population::Population, archive::Archive)
2           # Employee bee phase | 収穫蜂フェーズ
3           population, archive = employed_bee(population, archive)
4
5           # Onlooker bee phase | 追従蜂フェーズ
6           population, archive = onlooker_bee(population, archive)
7
8           # Scout bee phase | 偵察蜂フェーズ
9           population, archive = scout_bee(population, archive)
10
11          return population, archive  # 更新された個体群とアーカイブを返す
12      end
```

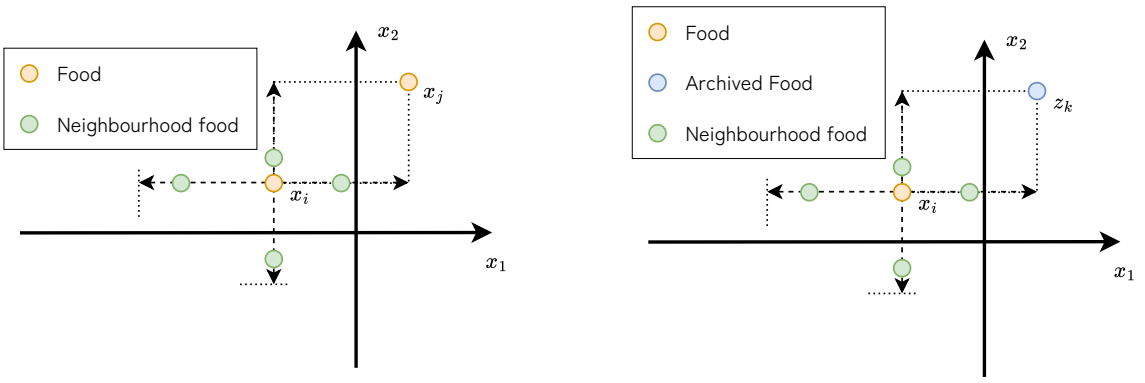

図 1: ABCME の探索の概要図 (左: 通常の探索，右: アーカイブを用いた探索)

ソースコード 22 は，ABC アルゴリズムの収穫蜂フェーズの処理を示している．まず，2 行目から 4 行目にかけて各変数を初期化する．次に，6 行目から 20 行目までの `for` 文は収穫蜂フェーズの処理を行う．7 行目から 17 行目の `for` 文は，各個体に対して次元ごとに変異ベクトルを計算する．8 行目から 14 行目では対象の個体とは異なるランダムな個体を選択する．この際，行われる探索は通常の ABC アルゴリズムと同様であり，その様子を図 1 の左図に示す．16 行目では変異ベクトルを計算し，19 行目で貪欲選択を行う．変異ベクトルの計算には，区間 $[-1, 1]$ の一様乱数関数 ϕ (ソースコード 23) を用いている．貪欲選択 (greedy selection) は，新しい解が良い場合は新しい解を採用し，悪い場合は元の解を採用する．ソースコード 24 の `greedySelection` 関数にその処理内容を記載する．3, 4 行目では，ベンチマーク値を計算し，6 行目から 12 行目では，新しい解が良い場合は試行回数をリセットし新しい解を返す．悪い場合は試行回数をインクリメントし，元の解を返す．

ソースコード 22: employed_bee 関数

```
1    function employed_bee(population::Population, archive::Archive)
2        I_P = population.individuals     # 個体群を取得
3        v_P = zeros(Float64, D)          # 変異ベクトルを初期化
4        j   = rand(RNG, 1:FOOD_SOURCE)   # ランダムなインデックスを生成
5
```

```
        for i in 1:FOOD_SOURCE
            for d in 1:D
                while true  # ランダムなインデックスを生成
                    j = rand(RNG, 1:FOOD_SOURCE)

                    if i != j
                        break
                    end
                end
                # 変異ベクトルを計算
                v_P[d] = I_P[i].genes[d] + ϕ() * (I_P[i].genes[d] - I_P[j].genes[d])
            end
            # 貪欲選択を行う
            population.individuals[i].genes = deepcopy(greedySelection(I_P[i].genes, v_P,
              trial_P, i))
        End

        return population, archive  # 更新された個体群とアーカイブを返す
    end
```

ソースコード 23: 一様乱数関数

```
    function ϕ()
        return rand(RNG) * 2.0 - 1.0  # 一様分布 [-1, 1]
    end
```

ソースコード 24: greedySelection 関数

```
    function greedySelection(x::Vector{Float64}, v::Vector{Float64}, trial::Vector{Int}, i
      ::Int)
        # ベンチマークを計算
        x_b = (objective_function(noise(x)), objective_function(x))
        v_b = (objective_function(noise(v)), objective_function(v))

        if fitness(v_b[fit_index]) > fitness(x_b[fit_index])  # 新しい解が良い場合
            trial[i] = 0  # 試行回数をリセット
            return v      # 新しい解を返す
        else
            trial[i] += 1  # 試行回数をインクリメント
            return x       # x の解を返す
        end
```

```
13      end
```

ソースコード 25 は，ABC アルゴリズムの追従蜂フェーズの処理を示している．まず，2 行目から 5 行目にかけて各変数を初期化する．7, 8 行目では，適応度の合計値を計算し，10, 11 行目では累積確率を計算する．13 行目から 39 行目までの `for` 文は追従蜂フェーズの処理を行う．まず，15 行目と 16 行目でルーレット選択を行い，18 行目から 24 行目で変異ベクトルを計算する．ルーレット選択を行う関数はソースコード??lst:roulette) の `rouletteSelection` 関数である．Julia 言語では多重ディスパッチを用いて，引数の型によって異なる関数を呼び出すことができる．そのため，`rouletteSelection` 関数は，Population と Archive の 2 つの関数を定義している．28, 29 行目では変異ベクトル `v_P`，`v_A` の評価を比較し，31 行目から 38 行目では貪欲選択を行う．この際行われる探索は，通常の探索 (図 1 の左図) と，アーカイブを用いた探索 (図 1 の右図) の両方が行われる．貪欲選択の処理内容は，収穫蜂フェーズと同様である．

ソースコード 25: onlooker_bee 関数

```
1   function onlooker_bee(population::Population, archive::Archive)
2       I_P, I_A = population.individuals, archive.individuals   # 個体群とアーカイブの個体を取得
3       v_P, v_A = zeros(Float64, D), zeros(Float64, D)          # 変異ベクトルを初期化
4       u_P, u_A = zeros(Float64, D), zeros(Float64, D)          # 交叉ベクトルを初期化
5       j, k     = rand(RNG, 1:FOOD_SOURCE), rand(RNG, collect(keys(I_A)))   # ランダムなインデックスを生成
6       # 適応度の合計を計算Σ
7       _fit_p = sum(fitness(I_P[i].benchmark[fit_index]) for i in 1:FOOD_SOURCE)Σ
8       _fit_a =sum(fitness(I_A[i].benchmark[fit_index]) for i in keys(I_A))
9       # 累積確率を計算
10      cum_p_p = [fitness(I_P[i].benchmark[fit_index]) / Σ_fit_p for i in 1:FOOD_SOURCE]
11      cum_p_a = Dict{Int64, Float64}(i => fitness(I_A[i].benchmark[fit_index]) / Σ_fit_a for i in keys(I_A))
12
13      for i in 1:FOOD_SOURCE
14          # ルーレット選択を行う
15          u_P = I_P[rouletteSelection(cum_p_p, I_P)].genes
16          u_A = I_A[rouletteSelection(cum_p_a, collect(keys(I_A)))].genes
17
```

```
            for d in 1:D
                while true  # ランダムなインデックスを生成
                    j, k = rand(RNG, 1:FOOD_SOURCE), rand(RNG, collect(keys(I_A)))

                    if I_P[i].genes[d] != I_A[k].genes[d] && i != j
                        break
                    end
                end

                # 変異ベクトルを計算
                v_P[d] = u_P[d] + ϕ() * (u_P[d] - I_A[k].genes[d])
                v_A[d] = u_A[d] + ϕ() * (u_A[d] - I_P[j].genes[d])
            end
            # 変異ベクトル v_P と v_A の評価を比較
            population.individuals[i].genes = if objective_function(v_P) <
              objective_function(v_A)
                # 個体 I_P[i] と変異ベクトル v_P とで貪欲選択を行う
                greedySelection(I_P[i].genes, v_P, trial_P, i)
            else
                # 個体 I_P[i] と変異ベクトル v_A とで貪欲選択を行う
                greedySelection(I_P[i].genes, v_A, trial_P, i)
            end
        end

        print(".")

        return population, archive  # 更新された個体群とアーカイブを返す
    end
```

ソースコード 26: rouletteSelection 関数

```
    # Roulette selection | Population
    function rouletteSelection(cum_probs::Vector{Float64}, I::Vector{Individual})
        r = rand(RNG)  # 乱数を生成

        for i in 1:length(I)
            if cum_probs[i] > r  # 累積確率が乱数よりも大きい場合
                return i  # 選択されたインデックスを返す
            end
        end

```

```
11          return rand(RNG, 1:length(I))  # ランダムなインデックスを返す
12      end
13
14      # Roulette selection | Archive
15      function rouletteSelection(cum_probs::Dict{Int64, Float64}, I::Vector{Int64})
16          r = rand(RNG)  # 乱数を生成
17
18          for key in I
19              if cum_probs[key] > r  # 累積確率が乱数よりも大きい場合
20                  return key  # 選択されたキーを返す
21              end
22          end
23
24          return rand(RNG, I)  # ランダムなキーを返す
25      end
```

ソースコード 27 は，ABC アルゴリズムの偵察蜂フェーズの処理を示している．このフェーズでは，試行回数が上限を超えた場合に新しい個体を生成とアーカイブの更新を行う．まず 4 行目では，施行回数を保存するベクトル `trial_P` の要素に上限値を超えたものがある場合の条件分岐を行っている．5 行目から 43 行目では，各個体に対して試行回数 `trial_P` が上限を超えた場合に新しい個体を生成する処理を行っている．8, 9 行目では個体の遺伝子を生成し，ノイズを加える．12 行目から 16 行目では，新しい遺伝子を元に新しい個体を生成する．17 行目では試行回数をリセットする．26 行目から 42 行目では，ボロノイ図の更新回数が上限値以下の場合に CVT を初期化し，新しいアーカイブを生成し，アーカイブを更新する処理を行っている．ただし，ボロノイ図の更新回数 `cvt_vorn_data_update` が上限値 `cvt_vorn_data_update_limit` を超えた場合はアーカイブの更新は行われない．アーカイブの更新は 28 行目の `init_CVT` 関数で行われる．31 行目から 35 行目では，新しいアーカイブを生成し，37 行目ではアーカイブを更新する．39 行目では，試行回数カウンタをリセットする．戻り値は，更新された `Population` 構造体と `Archive` 構造体である．

ソースコード 27: scout_bee 関数

```
1       function scout_bee(population::Population, archive::Archive)
```

```
        global trial_P, trial_A, cvt_vorn_data_update

        if maximum(trial_P) > TC_LIMIT  # 試行回数が上限を超えた場合
            for i in 1:FOOD_SOURCE
                if trial_P[i] > TC_LIMIT  # 試行回数が上限を超えた場合
                    # 新しい遺伝子を生成 -> ノイズを加える
                    gene        = rand(Float64, D) .* (UPP - LOW) .+ LOW
                    gene_noised = noise(gene)

                    # 新しい個体を生成
                    population.individuals[i] = Individual(
                        deepcopy(gene_noised),
                        (objective_function(gene_noised), objective_function(gene)),
                        devide_gene(gene_noised)
                    )
                    # 試行回数をリセット
                    trial_P[i] = 0

                    # 新しい食料源の発見をログに記録
                    logger("INFO", "Scout bee found a new food source")
                end
            end

            # ボロノイデータ更新回数が上限値以下の場合
            if cvt_vorn_data_update <= cvt_vorn_data_update_limit
                # を初期化CVT
                init_CVT(population)

                # 新しいアーカイブを生成
                new_archive = Archive(
                    zeros(Int64, 0, 0),
                    zeros(Int64, k_max),
                    Dict{Int64, Individual}()
                )
                # アーカイブを更新
                archive     = deepcopy(cvt_mapping(population, new_archive))
                # 試行回数カウンタをリセット
                trial_A     = zeros(Int, k_max)

                logger("INFO", "Recreate Voronoi diagram")  # ボロノイ図の再作成をログに記録
            end
        end
```

```
44
45          return population, archive   # 更新された個体群とアーカイブを返す
46      end
```

A　付録: GitHub リポジトリ

本研究で作成した ABE-MAP-Elites アルゴリズムのフローチャートは以下の通りである.

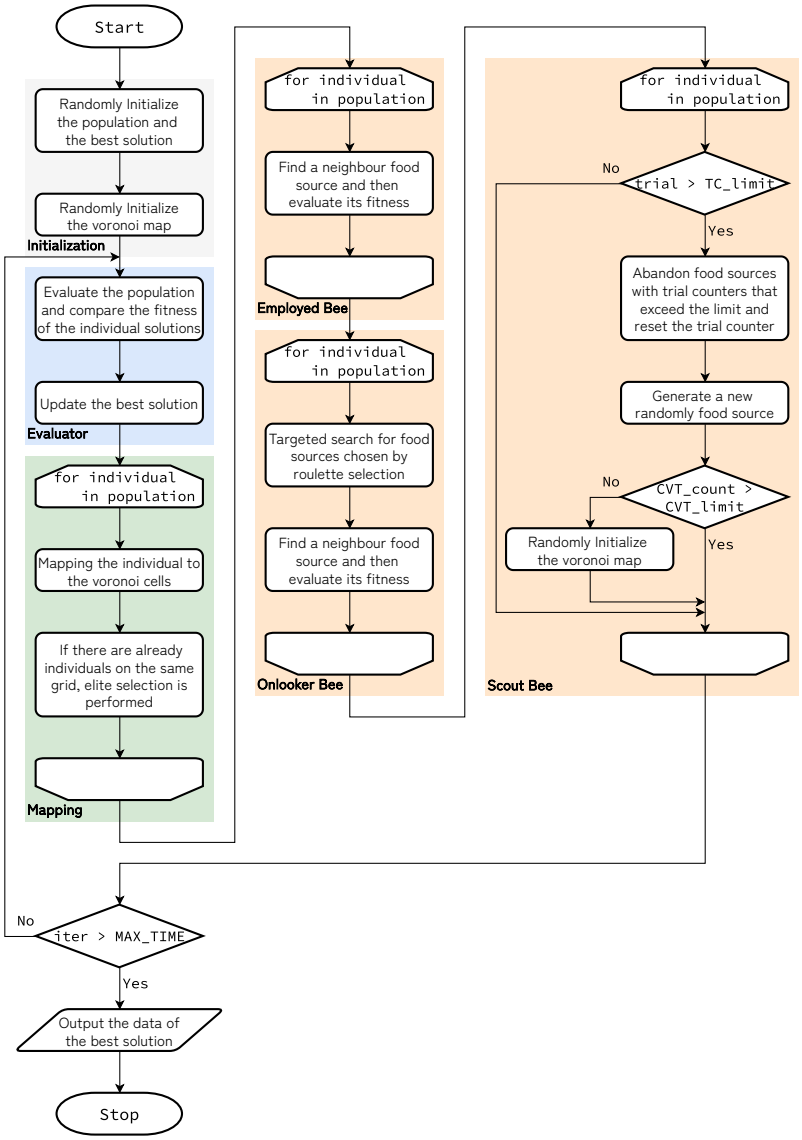

図 2: ABE-MAP-Elites アルゴリズムのフローチャート

B　付録: GitHub リポジトリ

本研究で作成及び使用したソースコードは以下のリンクからダウンロードできる.

- https://github.com/cyokozai/ABC_MAPElites

C　付録: ソースコード

ソースコード 28: main.jl

```
using Printf   # フォーマット付き文字列を出力

using Dates    # 日付と時間


include("config.jl")     # 設定ファイル

include("savedata.jl")   # データ保存用のファイル

include("me.jl")         # MAP-アルゴリズムの実装ファイルElites

include("logger.jl")     # ログ出力用のファイル


# Main
function main()
    # Make result directory and log file
    MakeFiles()

    CheckParameters()  # パラメータをチェック

    #------ MAP ELITES ALGORITHM ------------------------------#

    begin_time = time()                    # 開始時間を記録

    popn, arch, iter_time = map_elites()  # MAP-アルゴリズムを実行Elites

    finish_time = time()                   # 終了時間を記録

    #------ MAP ELITES ALGORITHM ------------------------------#

    elapsed_time = finish_time - begin_time   # 経過時間を計算

    println("End of Iteration.\n")
    println("Time of iteration: ", iter_time, " [sec]")      # 反復の時間を出力
    println("Time:               ", elapsed_time, " [sec]")  # 総経過時間を出力

```

```
38          # Save result
39          SaveResult(arch, iter_time, elapsed_time)
40      end
41
42
43      # Run
44      try
45          global exit_code = 0
46
47          logger("INFO", "Start")   # 開始ログを記録
48          println("Start")          # 開始メッセージを出力
49
50          main()   # メイン関数を実行
51
52          logger("INFO", "Success!␣:)")   # 成功ログを記録
53          println("Success!␣:)")          # 成功メッセージを出力
54      catch e
55          global exit_code = 1   # エラー発生時にをに設定exit_code1
56
57          logger("ERROR", "An␣error␣occurred!␣:(\n$e")   # エラーログを記録
58          println("An␣error␣occurred!␣:(\n$e")           # エラーメッセージを出力
59      finally
60          logger("INFO", "Finish")   # 終了ログを記録
61          println("Finish")          # 終了メッセージを出力
62
63          exit(exit_code)   # プログラムを終了
64      end
```

ソースコード 29: benchmark.jl

```
1       include("config.jl")   # 設定ファイル
2
3       include("logger.jl")   # ログ出力用のファイル
4
5
6       # Objective function
7       objective_function = if OBJ_F == "sphere"
8           # Sphere
9           x::Vector{Float64} -> sum(x .^ 2)
10      elseif OBJ_F == "rosenbrock"
11          # Rosenbrock
```

```
12          x::Vector{Float64} -> sum(100 .* (x[2:end] .- x[1:end-1].^2).^2 + (1 .- x[1:end-1])
                .^2)
13      elseif OBJ_F == "rastrigin"
14          # Rastrigin
15          x::Vector{Float64} -> sum(x .^ 2 - 10 * cos.(2 * pi * x) .+ 10)
16      elseif OBJ_F == "griewank"
17          # Griewank
18          x::Vector{Float64} -> sum(x .^ 2 / 4000) - prod(cos.(x ./ sqrt.(1:length(x)))) + 1
19      elseif OBJ_F == "ackley"
20          # Ackley
21          x::Vector{Float64} -> -20 * exp(-0.2 * sqrt(sum(x .^ 2) / length(x))) - exp(sum(cos
                .(2 * pi * x)) / length(x)) + 20 + exp(1)
22      elseif OBJ_F == "schwefel"
23          # Schwefel
24          x::Vector{Float64} -> 418.9829 * length(x) - sum(x .* sin.(sqrt.(abs.(x))))
25      else
26          logger("ERROR", "Objective␣function␣is␣invalid")  # 目的関数が無効であることをエラーログに記
                録 -> 終了
27
28          exit(1)
29      end
30
31
32      # Number of solution and bounds
33      SOLUTION, LOW, UPP = if OBJ_F == "sphere"
34          # Sphere
35          [zeros(D), -5.12, 5.12]
36      elseif OBJ_F == "rosenbrock"
37          # Rosenbrock
38          [zeros(D), -5.00, 5.00]
39      elseif OBJ_F == "rastrigin"
40          # Rastrigin
41          [zeros(D), -5.12, 5.12]
42      elseif OBJ_F == "griewank"
43          # Griewank
44          [zeros(D), -512.0, 512.0]
45      elseif OBJ_F == "ackley"
46          # Ackley
47          [zeros(D), -32.0, 32.0]
48      elseif OBJ_F == "schwefel"
49          # Schwefel
50          [zeros(D), -500.0, 500.0]
```

```
51      else
52          logger("ERROR", "Objective␣parameter␣is␣invalid")   # 目的関数パラメータが無効であることをエラーログに記録 -> 終了
53
54          exit(1)
55      end
```

ソースコード 30: config.jl

```
1       using StableRNGs   # 乱数生成
2
3       using Dates        # 日付と時間
4
5
6       # Method and Objective function
7       # Method: me, abc, de
8       const METHOD     = length(ARGS) > 1 ? ARGS[2] : "me"
9       if METHOD != "me" && METHOD != "abc" && METHOD != "de"
10          println("Error:␣The␣method␣is␣not␣available.")   # エラーログにメソッドが無効であることを記録 -> 終了
11
12          exit(1)
13      end
14
15      # MAP Method: grid, cvt
16      const MAP_METHOD = length(ARGS) > 2 ? ARGS[3] : "cvt"
17      if MAP_METHOD != "grid" && MAP_METHOD != "cvt"
18          println("Error:␣The␣MAP␣method␣is␣not␣available.")   # エラーログにマップメソッドが無効であることを記録 -> 終了
19
20          exit(1)
21      end
22
23      # Objective function: sphere, rosenbrock, rastrigin, griewank, ackley, schwefel, michalewicz
24      const OBJ_F      = length(ARGS) > 3 ? ARGS[4] : "sphere"
25      if OBJ_F != "sphere" && OBJ_F != "rosenbrock" && OBJ_F != "rastrigin" && OBJ_F != "griewank" && OBJ_F != "ackley" && OBJ_F != "schwefel" && OBJ_F != "michalewicz"
26          println("Error:␣The␣objective␣function␣is␣not␣available.")   # エラーログに目的関数が無効であることを記録 -> 終了
27
28          exit(1)
29      end
```

```
30
31
32      # Random Number Generator
33      # Random seed
34      SEED = Int(Dates.now().instant.periods.value)
35
36      # Random number generator
37      RNG  = StableRNG(SEED)
38
39
40      # Parameters
41      # Number of dimensions
42      const D          = length(ARGS) > 0 && ARGS[1] == "test" ? 2 : parse(Int64, ARGS[1])
43
44      # Number of population size | Default: 64
45      const N          = 64
46
47      # Dumber of behavior dimensions | Default: 2
48      const BD         = 2
49
50      # Convergence flag | 'true' is available when you want to check the convergence.
51      const CONV_FLAG = false
52
53      # Epsiron | Default: 1e-6
54      const EPS        = 1e-6
55
56      # Number of max time | Default: 100000
57      const MAXTIME    = if ARGS[1] == "test"  # テストの場合
58          1000
59      elseif CONV_FLAG == false  # 収束フラグが偽の場合
60          100000
61      else
62          println("Error: The objective function is not available.")  # エラーログに目的関数が無効であることを記録 -> 終了
63
64          exit(1)
65      end
66
67
68      # Noise parameter
69      # Fitness noise | 'true' is available when you want to add the noise to the fitness.
70      const FIT_NOISE = true
```

```
71
72      # Noise rate | Default: 0.01
73      const r_noise    = 0.01
74
75      # The number of mean gene
76      const MEAN_GENE = FIT_NOISE ? N : 1
77
78
79      # Map parameter
80      # MAP_METHOD == grid: Number of grid size. | Default: 158
81      const GRID_SIZE = 158
82
83      # MAP_METHOD == cvt: Number of max k. | Default: 25000
84      const k_max     = 25000
85
86      # CVT Max iteration | Default: 100
87      const CVT_MAX_ITER               = 100
88
89      # Voronoi data update limit | Default: 3
90      const cvt_vorn_data_update_limit = length(ARGS) > 4 ? parse(Int64, ARGS[5]) : 3
91
92
93      # MAP-Elites parameter
94      # Number of mutation rate | Default: 0.90
95      const MUTANT_R   = 0.90
96
97
98      # DE parameter
99      # The crossover probability & The differentiation (mutation) scaling factor | Default:
          0.80 & 0.90
100     const CR, F = if METHOD != "de"
101         [0.50, 0.00]
102     elseif OBJ_F == "sphere"
103         [0.20, 0.40]
104     elseif OBJ_F == "rosenbrock"
105         [0.70, 0.80]
106     elseif OBJ_F == "rastrigin"
107         [0.50, 0.60]
108     elseif OBJ_F == "griewank"
109         [0.40, 0.50]
110     elseif OBJ_F == "ackley"
111         [0.20, 0.50]
```

```
112     elseif OBJ_F == "schwefel"
113         [0.20, 0.50]
114     else
115         [0.80, 0.90]
116     end
117
118
119     # ABC parameter
120     # Food source: The number of limit trials that the employed bee can't find the better
          solution.
121     const FOOD_SOURCE = N
122
123     # Limit number: The number of limit trials that the scout bee can't find the better
          solution.
124     const TC_LIMIT    = D * floor(Int, k_max / (10 * FOOD_SOURCE))
125
126
127     # Result file
128     const output = "./result/"
129
130     if !isdir(output) || !isdir("$(output)$(METHOD)/$(OBJ_F)/") || !isdir("./log/")
131         mkpath(output)
132         mkpath("$(output)$(METHOD)/$(OBJ_F)/")
133         mkpath("./log/")
134     end
135
136     # Date
137     const DATE    = Dates.format(now(), "yyyy-mm-dd-HH-MM")
138     const LOGDATE = Dates.format(now(), "yyyy-mm-dd")
139
140     # File name
141     const FILENAME   = length(ARGS) > 0 && ARGS[1] == "test" ? "$(DATE)-test" : "$(METHOD)-
          $(MAP_METHOD)-$(OBJ_F)-$(D)-$(DATE)"
142     const F_RESULT   = "result-$(FILENAME).dat"
143     const F_FITNESS  = "fitness-$(FILENAME).dat"
144     const F_FIT_N    = "fitness-noise-$(FILENAME).dat"
145     const F_BEHAVIOR = "behavior-$(FILENAME).dat"
146     const F_LOGFILE  = "log-$(METHOD)-$(OBJ_F)-$(LOGDATE).log"
147
148     # EXIT CODE: 0: Success, 1: Failure
149     exit_code = 0
```

ソースコード 31: struct.jl

```
# Individual
mutable struct Individual
    # N dimension vector | 次元ベクトルN
    genes::Vector{Float64}

    # Benchmark value (1: with noise, 2: without noise) | ベンチマーク値 (1: ノイズあり, 2: ノイズなし)
    benchmark::Tuple{Float64, Float64}

    # Behavior space | 行動空間
    behavior::Vector{Float64}
end


# Population
mutable struct Population
    # Group of individuals | 個体群
    individuals::Vector{Individual}
end


# Archive
mutable struct Archive
    # Grid map | グリッドマップ
    grid::Matrix{Int64}

    # Grid update counts | グリッド更新回数
    grid_update_counts::Vector{Int64}

    # Individuals | 個体
    individuals::Dict{Int64, Individual}
end
```

ソースコード 32: logger.jl

```
using Dates  # 日付と時間


# Logger
function logger(status::String, message::String)
```

```
6         open("log/$F_LOGFILE", "a") do fl # ログファイルを開く
7             println(fl, Dates.format(now(), "yyyy-mm-dd␣HH:MM:SS"), "[", status, "]␣",
                message) # ログを出
                力
8         end
9     end
```

ソースコード 33: savedata.jl

```
1     using Printf   # フォーマット付き文字列を出力
2
3     using Dates    # 日付と時間
4
5
6     include("config.jl")     # 設定ファイル
7
8     include("struct.jl")     # 構造体
9
10    include("fitness.jl")    # 適応度
11
12    include("cvt.jl")        # 関連のファイルCVT
13
14    include("logger.jl")     # ログ出力用のファイル
15
16
17    # Make result directory and log file
18    function MakeFiles()
19        open("$(output)$METHOD/$OBJ_F/$F_RESULT", "w") do fr
20            println(fr, "Date:␣", DATE)
21            println(fr, "Method:␣", METHOD)
22            if METHOD == "de"
23                println(fr, "F:␣", F)
24                println(fr, "CR:␣", CR)
25            elseif METHOD == "abc"
26                println(fr, "Trial␣count␣limit:␣", TC_LIMIT)
27            end
28            println(fr, "Map:␣", MAP_METHOD)
29            if MAP_METHOD == "cvt"
30                println(fr, "Voronoi␣point:␣", k_max)
31            end
32            println(fr, "Noise:␣", FIT_NOISE)
33            println(fr, "Benchmark:␣", OBJ_F)
```

```
34              println(fr, "Dimension:␣", D)
35              println(fr, "Population␣size:␣", N)
36          end
37
38          if FIT_NOISE  # ノイズがある場合
39              open("$(output)$METHOD/$OBJ_F/$F_FIT_N", "w") do ffn  # fitness.dat ファイルを開く
40                  println(ffn, "Date:␣", DATE)
41                  println(ffn, "Method:␣", METHOD)
42                  if METHOD == "de"
43                      println(ffn, "F:␣", F)
44                      println(ffn, "CR:␣", CR)
45                  elseif METHOD == "abc"
46                      println(ffn, "Trial␣count␣limit:␣", TC_LIMIT)
47                  end
48                  println(ffn, "Map:␣", MAP_METHOD)
49                  if MAP_METHOD == "cvt"
50                      println(ffn, "Voronoi␣point:␣", k_max)
51                  end
52                  println(ffn, "Noise:␣", FIT_NOISE)
53                  println(ffn, "Benchmark:␣", OBJ_F)
54                  println(ffn, "Dimension:␣", D)
55                  println(ffn, "Population␣size:␣", N)
56              end
57          end
58
59          open("$(output)$METHOD/$OBJ_F/$F_FITNESS", "w") do ff  # futness.dat ファイルを開く
60              println(ff, "Date:␣", DATE)
61              println(ff, "Method:␣", METHOD)
62
63              if METHOD == "de"  #DME
64                  println(ff, "F:␣", F)
65                  println(ff, "CR:␣", CR)
66              elseif METHOD == "abc"  #ABCME
67                  println(ff, "Trial␣count␣limit:␣", TC_LIMIT)
68              end
69
70              println(ff, "Map:␣", MAP_METHOD)
71
72              if MAP_METHOD == "cvt"  # マップの場合CVT
73                  println(ff, "Voronoi␣point:␣", k_max)
74              end
75
```

```
76              println(ff, "Noise: ", FIT_NOISE)
77              println(ff, "Benchmark: ", OBJ_F)
78              println(ff, "Dimension: ", D)
79              println(ff, "Population size: ", N)
80          end
81
82          open("$(output)$METHOD/$OBJ_F/$F_BEHAVIOR", "w") do fb
83              println(fb, "Date: ", DATE)
84              println(fb, "Method: ", METHOD)
85              if METHOD == "DE"  #DME
86                  println(fb, "F: ", F)
87                  println(fb, "CR: ", CR)
88              elseif METHOD == "ABC"  #ABCME
89                  println(fb, "Trial count limit: ", TC_LIMIT)
90              end
91
92              println(fb, "Map: ", MAP_METHOD)
93
94              if MAP_METHOD == "cvt"  # マップの場合CVT
95                  println(fb, "Voronoi point: ", k_max)
96              end
97
98              println(fb, "Noise: ", FIT_NOISE)
99              println(fb, "Benchmark: ", OBJ_F)
100             println(fb, "Dimension: ", D)
101             println(fb, "Population size: ", N)
102         end
103     end
104
105
106     # Check the parameters
107     function CheckParameters()
108         # Check dimension
109         if D == 2
110             logger("WARN", "Dimension is default value \"2\"")  # 次元がデフォルト値「」であること
                を警告2
111         elseif D <= 0
112             logger("ERROR", "Dimension is invalid")  # 次元が無効であることをエラーログに記録 -> 終了
113
114             exit(1)
115         else
116             logger("INFO", "Dimension is $D")  # 次元を情報ログに記録
```

```
117         end
118
119         # Convergence mode check
120         if CONV_FLAG
121             logger("INFO", "Convergence␣flag␣is␣true")    # 収束フラグが真であることを情報ログに記録
122         else
123             logger("INFO", "Convergence␣flag␣is␣false")   # 収束フラグが偽であることを情報ログに記録
124         end
125
126         # Check method
127         println("Method␣␣␣:␣", METHOD)  # 使用するメソッドを出力
128
129         if METHOD == "de"
130             println("␣␣␣␣␣␣␣F␣:␣", F)    # 差分進化の値を出力F
131             println("␣␣␣␣␣␣␣CR:␣", CR)   # 差分進化の交叉率を出力
132         elseif METHOD == "abc"
133             println("Trial␣count␣limit:␣", TC_LIMIT)  # アルゴリズムの試行回数制限を出力ABC
134         end
135
136         println("Map␣␣␣␣␣␣:␣", MAP_METHOD)  # マップメソッドを出力
137         println("Voronoi␣point:␣", k_max)       # ボロノイ点の数を出力
138
139         # Print parameters
140         println("Benchmark:␣", OBJ_F)     # ベンチマーク関数を出力
141         println("Dimension:␣", D)         # 次元を出力
142         println("Population␣size:␣", N)   # 集団サイズを出力
143
144     end
145
146
147     # Save result
148     function SaveResult(archive::Archive, iter_time::Float64, run_time::Float64)
149         # Log file
150         logger("INFO", "Time␣of␣iteration:␣$iter_time␣[sec]")
151         logger("INFO", "Time:␣$run_time␣[sec]")
152
153         # Open file
154         ffn = open("$(output)$METHOD/$OBJ_F/$F_FIT_N", "a")
155         fr  = open("$(output)$METHOD/$OBJ_F/$F_RESULT", "a")
156         ff  = open("$(output)$METHOD/$OBJ_F/$F_FITNESS", "a")
157         fb  = open("$(output)$METHOD/$OBJ_F/$F_BEHAVIOR", "a")
158
```

```
159         if MAP_METHOD == "cvt"  # マップの場合CVT
160             for (k, v) in archive.individuals
161                 println(ffn, archive.individuals[k].benchmark[1])
162                 println(ff, archive.individuals[k].benchmark[2])
163                 println(fb, archive.individuals[k].behavior)
164                 println(fr, archive.grid_update_counts[k])
165             end
166         else
167             logger("ERROR", "Map method is invalid")  # マップメソッドが無効であることをエラーログに記録 -> 終了
168 
169             exit(1)
170         end
171 
172         # Close file
173         if FIT_NOISE
174             close(ffn)
175             close(fr)
176             close(ff)
177             close(fb)
178         end
179 
180         logger("INFO", "End of Iteration")  # 反復終了のログを記録
181 
182         # Make result list
183         arch_list = []
184 
185         if MAP_METHOD == "cvt"  # マップの場合CVT
186             for k in keys(archive.individuals)
187                 if k > 0  # インデックスがより大きい場合0
188                     push!(arch_list, archive.individuals[k])
189                 end
190             end
191         else
192             logger("ERROR", "Map method is invalid")  # マップメソッドが無効であることをエラーログに記録 -> 終了
193 
194             exit(1)
195         end
196 
197         sort!(arch_list, by = x -> fitness(x.benchmark[fit_index]), rev = true)  # 適応度でソート
198 
```

```
199         open("$(output)$METHOD/$OBJ_F/$F_RESULT", "a") do fr  # 結果ファイルを開く
200             println(fr, "End␣of␣Iteration.\n")
201             println(fr, "Time␣of␣iteration:␣", iter_time, "␣[sec]")
202             println(fr, "Time:␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣", run_time, "␣[sec]")
203             println(fr, "The␣number␣of␣solutions:␣", length(arch_list))
204             println(fr, "The␣number␣of␣regenerated␣CVT␣Map:␣", cvt_vorn_data_update)
205             println(fr, "Top␣10␣suboptimal␣solutions:")
206 
207             for i in 1:10
208                 println(fr, "Rank␣", i, ":␣")
209                 println(fr, "├──␣Solution:␣␣␣␣␣␣", arch_list[i].genes)
210                 println(fr, "├──␣Noisy␣Fitness:␣", fitness(arch_list[i].benchmark[1]))
211                 println(fr, "├──␣True␣Fitness:␣␣", fitness(arch_list[i].benchmark[2]))
212                 println(fr, "└──␣Behavior:␣␣␣␣␣␣", arch_list[i].behavior)
213             end
214         end
215 
216 
217         println("Best␣solution:␣␣␣␣␣␣", best_solution.genes)
218         println("Best␣noisy␣fitness:␣", fitness(best_solution.benchmark[1]))
219         println("Best␣true␣fitness:␣␣", fitness(best_solution.benchmark[2]))
220         println("Best␣behavior:␣␣␣␣␣␣", best_solution.behavior)
221 
222     end
```

ソースコード 34: cvt.jl

```
1     using DelaunayTriangulation  # 三角形分割Delaunay
2 
3     using LinearAlgebra          # 線形代数
4 
5     using StableRNGs             # 乱数生成
6 
7     using FileIO                 # ファイル入出力
8 
9     using JLD2                   # ファイルJLD2
10 
11    using Dates                  # 日付と時間
12 
13 
14    include("config.jl")  # 設定ファイル
```

```
15
16      include("struct.jl")   # 構造体
17
18      include("logger.jl")   # ログ出力用のファイル
19
20
21      # Voronoi diagram
22      vorn = nothing              # ボロノイ図の初期化
23
24      # Voronoi data update
25      cvt_vorn_data_update = 0   # ボロノイデータ更新カウンタの初期化
26
27
28      # Initialize the CVT
29      function init_CVT(population::Population)
30          global vorn, cvt_vorn_data_update
31
32          points    = [rand(RNG, BD) .* (UPP - LOW) .+ LOW for _ in 1:k_max - (N + 4)]   # ランダムな点を生成
33          behavior = [population.individuals[i].behavior for i in 1:N]                    # 個体の行動を取得
34
35          append!(points, behavior)                                                       # 行動を点に追加
36          append!(points, [[UPP, UPP], [UPP, LOW], [LOW, UPP], [LOW, LOW]])   # 境界点を追加
37
38          vorn = centroidal_smooth(voronoi(triangulate(points; rng = RNG), clip = false); maxiters = CVT_MAX_ITER, rng = RNG)   # ボロノイ図を生成
39          save("$(output)$(METHOD)/$(OBJ_F)/CVT-$(FILENAME)-$(cvt_vorn_data_update).jld2", "voronoi", vorn)                      # ボロノイ図を保存
40
41          cvt_vorn_data_update += 1   # 更新カウンタをインクリメント
42
43          logger("INFO", "CVT␣is␣initialized")   # 初期化完了のログを記録
44
45          return DelaunayTriangulation.get_generators(vorn)::Dict{Int64, Tuple{Float64, Float64}}   # ボロノイ生成点を返す
46      end
47
48
49      # CVT mapping
50      function cvt_mapping(population::Population, archive::Archive)
```

```
51          global vorn
52
53          for ind in population.individuals
54              distances = [norm([ind.behavior[1] - centroid[1], ind.behavior[2] - centroid
                  [2]], 2) for centroid in values(DelaunayTriangulation.get_generators(vorn))]
                   # 各生成点との距離を
                  計算
55
56              closest_centroid_index = argmin(distances)  # 最も近い生成点のインデックスを取得
57
58              if haskey(archive.individuals, closest_centroid_index)  # アーカイブに生成点が存在する
                  場合
59                  if ind.benchmark[fit_index] < archive.individuals[closest_centroid_index].
                      benchmark[fit_index]  # 個体の評価がアーカイブよりも良い
                      場合
60                      archive.individuals[closest_centroid_index] = Individual(deepcopy(ind.
                          genes), ind.benchmark, deepcopy(ind.behavior))  # アーカイブを
                          更新
61
62                      archive.grid_update_counts[closest_centroid_index] += 1  # 更新カウンタをイ
                          ンクリメント
63                  end
64              else  # アーカイブに生成点が存在しない場合
65                  archive.individuals[closest_centroid_index] = Individual(deepcopy(ind.genes
                      ), ind.benchmark, deepcopy(ind.behavior))  # 新しい個体をアーカイブに
                      追加
66
67                  archive.grid_update_counts[closest_centroid_index] += 1  # 更新カウンタをインク
                      リメント
68              end
69          end
70
71          return archive  # 更新されたアーカイブを返す
72      end
```

ソースコード 35: fitness.jl

```
1       using Random          # 乱数生成
2
3       using Distributions   # 確率分布
4
5
6       include("benchmark.jl")  # ベンチマーク関数
7
8       include("config.jl")     # 設定ファイル
```

```
9
10
11      # Noise setting
12      # The flag of the fitness value. | 'true' is available when you want to add the noise
          to the fitness.
13      const fit_index = FIT_NOISE ? 1 : 2
14
15      # Constant of the noise. | Noise range is [-r_noise, r_noise]
16      const σ = r_noise / 4.0
17
18      # Normal distribution for the noise.
19      const N_noise = Normal(0.0, σ^(2.0))
20
21
22      # Fitness function
23      fitness(x::Float64) = x >= 0 ?  1.0 / (1.0 + x) : 1.0 + abs(x)
24
25
26      # Noise function with Gaussian function
27      noise(gene::Vector{Float64}) = FIT_NOISE ? [sum(x + rand(RNG, N_noise) for _ in 1:
          MEAN_GENE) / MEAN_GENE for x in gene] : gene
```

ソースコード 36: crossover.jl

```
1       using Random  # 乱数生成
2
3
4       include("config.jl")  # 設定ファイル
5
6
7       # Binominal crossover | DE/rand/1/bin
8       crossover(x::Vector{Float64}, v::Vector{Float64}) = [d == rand(RNG, 1:D) || rand(RNG) <
            CR ? v[d] : x[d] for d in 1:D]
9
10
11      # Uniform crossover
12      crossover(ind::Tuple{Individual, Individual}) = Individual(crossover(ind[1].genes, ind
          [2].genes), (0.0, 0.0), zeros(Float64, BD))
```

ソースコード 37: me.jl

```
using StableRNGs  # 安定した乱数生成

using Random      # 乱数生成


include("struct.jl")     # 構造体

include("config.jl")     # 設定ファイル

include("benchmark.jl")  # ベンチマーク関数

include("fitness.jl")    # 適応度

include("crossover.jl")  # 交叉

include("cvt.jl")        # 関連のファイルCVT

include("abc.jl")        # アルゴリズムABC

include("de.jl")         # 差分進化アルゴリズム

include("logger.jl")     # ログ出力用のファイル


# Devide gene
function devide_gene(gene::Vector{Float64})
    gene_len    = length(gene)        # 遺伝子の長さを取得
    segment_len = div(gene_len, BD)   # セグメントの長さを計算
    behavior    = Float64[]           # 行動ベクトルを初期化

    for i in 1:BD
        start_idx = (i - 1) * segment_len + 1             # 開始インデックスを計算
        end_idx   = i == BD ? gene_len : i * segment_len  # 終了インデックスを計算

        push!(behavior, BD*sum(gene[start_idx:end_idx])/Float64(gene_len))  # 行動ベクトルに値を追加
    end

    return behavior  # 行動ベクトルを返す
end


```

```
42      # Initialize the solutions
43      function init_solution()
44          gene = rand(RNG, D) .* (UPP - LOW) .+ LOW   # ランダムな遺伝子を生成
45          gene_noised = noise(gene)                   # ノイズを加える
46
47          return Individual(deepcopy(gene_noised), (objective_function(gene_noised),
              objective_function(gene)), devide_gene(gene_noised))   # 個体を生成して
              返す
48      end
49
50
51      # Best solution
52      best_solution = init_solution()   # 最良解を初期化
53
54
55      # Evaluator: Evaluation of the individual
56      function evaluator(individual::Individual)
57          global best_solution
58
59          # Objective function
60          gene_noised = noise(individual.genes)   # ノイズを加える
61          individual.benchmark = (objective_function(gene_noised), objective_function(
              individual.genes))   # ベンチマークを計算ノイズ付きとノイズなし
               ()
62
63          # Evaluate the behavior
64          individual.behavior = deepcopy(devide_gene(gene_noised))   # 行動を評価
65
66          # Update the best solution
67          if individual.benchmark[fit_index] <= best_solution.benchmark[fit_index]   # 最良解よ
              り個体の評価が良い場合、最良解を更新
68              best_solution = Individual(deepcopy(individual.genes), deepcopy(individual.
                  benchmark), deepcopy(individual.behavior))
69          end
70
71          return individual   # 評価された個体を返す
72      end
73
74
75      # Mapping: Mapping the individual to the archive
76      Mapping = if MAP_METHOD == "cvt"
77          (population::Population, archive::Archive) -> cvt_mapping(population, archive)   #
              マッピン
              グCVT
```

```
78      else
79          error("Invalid␣MAP␣method")  # 無効なメソッドエラーMAP -> 終了
80
81          logger("ERROR", "Invalid␣MAP␣method")  # ログ出力
82
83          exit(1)
84      end
85
86
87      # Mutate: Mutation of the individual
88      mutate(individual::Individual) = Individual([rand(RNG) < MUTANT_R ? rand(RNG) * (UPP -
          LOW) + LOW : gene for gene in individual.genes], (0.0, 0.0), zeros(Float64, BD))
89
90
91      # Select random elite
92      select_random_elite = if MAP_METHOD == "cvt"
93          (population::Population, archive::Archive) -> begin
94              random_centroid_index = zeros(Int64)  # ランダムなインデックスを初期化
95
96              return archive.individuals[random_centroid_index]  # ランダムなエリートを選択
97          end
98      end
99
100
101     # Reproduction: Generate new individuals
102     Reproduction = if METHOD == "me"
103         # MAP-Elites
104         (population::Population, archive::Archive) -> (Population([evaluator(mutate(
              select_random_elite(population, archive))) for _ in 1:N]), archive)
105     elseif METHOD == "de"
106         # Differential MAP-Elites
107         (population::Population, archive::Archive) -> DE(population, archive)
108     elseif METHOD == "abc"
109         # ABC MAP-Elites
110         (population::Population, archive::Archive) -> ABC(population, archive)
111     else
112         error("Invalid␣method")  # 無効なメソッドエラー -> 終了
113
114         logger("ERROR", "Invalid␣method")  # ログ出力
115
116         exit(1)
117     end
```

```
118
119
120     # Map Elites algorithm
121     function map_elites()
122         global best_solution
123
124         # Print the solutions
125         indPrint = if FIT_NOISE
126             (ffn, ff) -> begin
127                 println("Now␣best␣individual:␣", best_solution.genes[1:min(10, length(
                      best_solution.genes))])  # 最良個体を出力ただし、最大個まで(10)
128                 println("Now␣best␣behavior:␣␣␣", best_solution.behavior)
                                                          # 最良行動を出力
129                 println("Now␣noised␣␣␣␣best␣fitness:␣", fitness(best_solution.benchmark[1])
                    )                     # ノイズ付き最良適応度を出力
130                 println("Now␣corrected␣best␣fitness:␣", fitness(best_solution.benchmark[2])
                    )                     # ノイズなし最良適応度を出力
131
132                 println(ffn, best_solution.benchmark[1])  # ノイズ付き最良適応度をファイルに出力
133                 println(ff, best_solution.benchmark[2])   # ノイズなし最良適応度をファイルに出力
134             end
135         end
136
137         #------ Initialize ------------------------------#
138
139         logger("INFO", "Initialize")
140
141         # Initialize the population
142         population::Population = Population([evaluator(init_solution()) for _ in 1:N])  #
              個体群を初期化
143
144         # Initialize the archive
145         archive::Archive = if MAP_METHOD == "cvt"  # マッピングの場合CVT
146             init_CVT(population)  # を初期化CVT
147             Archive(zeros(Int64, 0, 0), zeros(Int64, k_max), Dict{Int64, Individual}())  #
                  アーカイブを初期化
148         else
149             error("Invalid␣MAP␣method")  # 無効なメソッドエラーMAP -> 終了
150
```

```
151             logger("ERROR", "Invalid␣MAP␣method")  # ログ出力
152
153             exit(1)
154         end
155
156         #------ Initialize ----------------------------#
157         # Open file
158         ffn = open("$(output)$(METHOD)/$(OBJ_F)/$(F_FIT_N)", "a")
159         ff  = open("$(output)$(METHOD)/$(OBJ_F)/$(F_FITNESS)", "a")
160
161         #------ Main loop ----------------------------#
162
163         logger("INFO", "Start␣Iteration")
164
165         begin_time = time()  # 初期化後の開始時間を取得
166
167         for iter in 1:MAXTIME  # Iteration for MAXTIME
168             # Print the generation
169             println("Generation:␣", iter)
170
171             # Evaluator
172             population = Population([evaluator(ind) for ind in population.individuals])
173
174             # Mapping
175             archive = Mapping(population, archive)
176
177             # Reproduction
178             population, archive = Reproduction(population, archive)
179
180             # Print the solutions
181             indPrint(ffn, ff)
182         end
183
184         finish_time = time()  # 終了時間を取得
185
186         logger("INFO", "Time␣out")
187
188         #------ Main loop ----------------------------#
189         # Close file
190         close(ffn)
191         close(ff)
192
```

```
193         return population, archive, (finish_time - begin_time)  # 更新された個体群とアーカイブ、経過時間を返す
194     end
```

ソースコード 38: de.jl

```
1     using Statistics  # 統計関数
2
3     using Random      # 乱数生成
4
5
6     include("config.jl")      # 設定ファイル
7
8     include("struct.jl")      # 構造体
9
10    include("fitness.jl")     # 適応度
11
12    include("crossover.jl")   # 交叉
13
14    include("logger.jl")      # ログ出力用のファイル
15
16
17    # Differential Evolution algorithm
18    function DE(population::Population, archive::Archive)
19        I_p, I_a = population.individuals, archive.individuals  # 個体群とアーカイブの個体を取得
20        r1, r2, r3 = zeros(Int, 3)                              # ランダムなインデックスを初期化
21        b = Tuple{Float64, Float64}[]                           # ベンチマーク結果を格納するタプルの配列を初期化
22
23        print("DE")
24
25        for i in 1:N
26            while r1 == r2 || r1 == r3 || r2 == r3 || I_a[r1].genes == I_p[i].genes || I_a[r2].genes == I_p[i].genes || I_a[r3].genes == I_p[i].genes
27                r1, r2, r3 = rand(RNG, keys(I_a), 3)  # ランダムな異なるインデックスを生成 -> ドナーベクトルを選択
28            end
29
30            v = clamp.(I_a[r1].genes .+ F .* (I_a[r2].genes .- I_a[r3].genes), LOW, UPP)  # 差分ベクトルを計算
31            u = crossover(I_p[i].genes, v)  # 二項交叉を行い、トライアルベクトルを計算
32
```

```
33              u_noised = noise(u)  # ノイズを加える
34              b = (objective_function(u_noised), objective_function(u))  # ベンチマークを計算
35
36              if b[fit_index] < I_a[r1].benchmark[fit_index]  # ターゲットベクトルとドナーベクトルの評
                  価を比較r1
37                  archive.individuals[r1] = Individual(deepcopy(u), b, devide_gene(u))    #
                      アーカイブを更
                      新r1
38              end
39
40              if b[fit_index] < I_a[r2].benchmark[fit_index]  # ターゲットベクトルとドナーベクトルの評
                  価を比較r2
41                  archive.individuals[r2] = Individual(deepcopy(u), b, devide_gene(u))    #
                      アーカイブを更
                      新r2
42              end
43
44              if b[fit_index] < I_a[r3].benchmark[fit_index]  # ターゲットベクトルとドナーベクトルの評
                  価を比較r3
45                  archive.individuals[r3] = Individual(deepcopy(u), b, devide_gene(u))    #
                      アーカイブを更
                      新r3
46              end
47
48              if b[fit_index] < I_p[i].benchmark[fit_index]  # ターゲットベクトルとトライアルベクトルの
                  評価を比較
49                  population.individuals[i] = Individual(deepcopy(u), b, devide_gene(u))  #
                      個体群を更
                      新
50              end
51          end
52
53          println("␣done")
54
55          return population, archive  # 更新された個体群とアーカイブを返す
56      end
```

ソースコード 39: abc.jl

```
1       using Distributions  # 分布関数
2
3       using Random         # 乱数生成
4
5
6       include("config.jl")     # 設定ファイル
```

```
7
8       include("struct.jl")      # 構造体
9
10      include("fitness.jl")     # 適応度
11
12      include("crossover.jl")   # 交叉
13
14      include("logger.jl")      # ログ出力用のファイル
15
16
17      # Trial counter | Population
18      trial_P = zeros(Int, FOOD_SOURCE)   # 試行回数カウンタ（個体群）
19
20      # Trial counter | Archive
21      trial_A = zeros(Int, k_max)         # 試行回数カウンタ（アーカイブ）
22
23
24      # Uniform distribution | [-1, 1]
25      function ϕ()
26          return rand(RNG) * 2.0 - 1.0   # 一様分布 [-1, 1]
27      end
28
29
30      # Greedy selection
31      function greedySelection(x::Vector{Float64}, v::Vector{Float64}, trial::Vector{Int}, i
          ::Int)
32          x_b, v_b = (objective_function(noise(x)), objective_function(x)), (
              objective_function(noise(v)), objective_function(v))   # ベンチマークを
              計算
33
34          if fitness(v_b[fit_index]) > fitness(x_b[fit_index])   # 新しい解が良い場合
35              trial[i] = 0   # 試行回数をリセット
36
37              return v       # 新しい解を返す
38          else
39              trial[i] += 1   # 試行回数をインクリメント
40
41              return x        # 元の解を返す
42          end
43      end
44
45
```

```
46      # Roulette selection | Population
47      function rouletteSelection(cum_probs::Vector{Float64}, I::Vector{Individual})
48          r = rand(RNG)  # 乱数を生成
49
50          for i in 1:length(I)
51              if cum_probs[i] > r  # 累積確率が乱数よりも大きい場合
52                  return i  # 選択されたインデックスを返す
53              end
54          end
55
56          return rand(RNG, 1:length(I))  # ランダムなインデックスを返す
57      end
58
59
60      # Roulette selection | Archive
61      function rouletteSelection(cum_probs::Dict{Int64, Float64}, I::Vector{Int64})
62          r = rand(RNG)  # 乱数を生成
63
64          for key in I
65              if cum_probs[key] > r  # 累積確率が乱数よりも大きい場合
66                  return key  # 選択されたキーを返す
67              end
68          end
69
70          return rand(RNG, I)  # ランダムなキーを返す
71      end
72
73
74      # Employed bee phase
75      function employed_bee(population::Population, archive::Archive)
76          I_P = population.individuals     # 個体群を取得
77          v_P = zeros(Float64, D)          # 変異ベクトルを初期化
78          j   = rand(RNG, 1:FOOD_SOURCE)  # ランダムなインデックスを生成
79
80          print(".")
81
82          for i in 1:FOOD_SOURCE
83              for d in 1:D
84                  while true
85                      j = rand(RNG, 1:FOOD_SOURCE)  # ランダムなインデックスを生成
86
87                      if i != j
```

```
88                          break
89                      end
90                  end
91
92                  v_P[d] = I_P[i].genes[d] + ϕ() * (I_P[i].genes[d] - I_P[j].genes[d])  # 変異ベクトルを計算
93              end
94
95              population.individuals[i].genes = deepcopy(greedySelection(I_P[i].genes, v_P, trial_P, i))  # 貪欲選択を行う
96          end
97
98          print(".")
99
100         return population, archive  # 更新された個体群とアーカイブを返す
101     end
102
103
104     # Onlooker bee phase
105     function onlooker_bee(population::Population, archive::Archive)
106         I_P, I_A = population.individuals, archive.individuals      # 個体群とアーカイブの個体を取得
107         v_P, v_A = zeros(Float64, D), zeros(Float64, D)             # 変異ベクトルを初期化
108         u_P, u_A = zeros(Float64, D), zeros(Float64, D)             # 交叉ベクトルを初期化
109         j, k     = rand(RNG, 1:FOOD_SOURCE), rand(RNG, collect(keys(I_A)))  # ランダムなインデックスを生成Σ
110
111         _fit_p, Σ_fit_a = sum(fitness(I_P[i].benchmark[fit_index]) for i in 1:FOOD_SOURCE), sum(fitness(I_A[i].benchmark[fit_index]) for i in keys(I_A))  # 適応度の合計を計算
112         cum_p_p, cum_p_a = [fitness(I_P[i].benchmark[fit_index]) / Σ_fit_p for i in 1:FOOD_SOURCE], Dict{Int64, Float64}(i => fitness(I_A[i].benchmark[fit_index]) / Σ_fit_a for i in keys(I_A))  # 累積確率を計算
113
114         print(".")
115
116         for i in 1:FOOD_SOURCE
117             u_P, u_A = I_P[rouletteSelection(cum_p_p, I_P)].genes, I_A[rouletteSelection(cum_p_a, collect(keys(I_A)))].genes  # ルーレット選択を行う
118
119             for d in 1:D
120                 while true
121                     j, k = rand(RNG, 1:FOOD_SOURCE), rand(RNG, collect(keys(I_A)))  # ランダムなインデックスを生成
```

```
122
123                     if I_P[i].genes[d] != I_A[k].genes[d] && i != j
124                         break
125                     end
126                 end
127
128                 v_P[d], v_A[d] = u_P[d] + ϕ() * (u_P[d] - I_A[k].genes[d]), u_A[d] + ϕ
                      () * (u_A[d] - I_P[j].genes[d])  # 変異ベクトルを計算
129             end
130
131             population.individuals[i].genes = if objective_function(v_P) <
                  objective_function(v_A)  # 変異ベクトルとの評価を比
                  較v_Pv_A
132                 greedySelection(I_P[i].genes, v_P, trial_P, i)  # 個体I_P[iと変異ベクトル]とで貪
                      欲選択を行うv_P
133             else
134                 greedySelection(I_P[i].genes, v_A, trial_P, i)  # 個体I_P[iと変異ベクトル]とで貪
                      欲選択を行うv_A
135             end
136         end
137
138         print(".")
139
140         return population, archive  # 更新された個体群とアーカイブを返す
141     end
142
143
144     # Scout bee phase
145     function scout_bee(population::Population, archive::Archive)
146         global trial_P, trial_A, cvt_vorn_data_update
147
148         print(".")
149
150         if maximum(trial_P) > TC_LIMIT  # 試行回数が上限を超えた場合
151             for i in 1:FOOD_SOURCE
152                 if trial_P[i] > TC_LIMIT  # 試行回数が上限を超えた場合
153                     gene        = rand(Float64, D) .* (UPP - LOW) .+ LOW  # 新しい遺伝子を生成
154                     gene_noised = noise(gene)  # ノイズを加える
155
156                     population.individuals[i] = Individual(deepcopy(gene_noised), (
                          objective_function(gene_noised), objective_function(gene)),
                          devide_gene(gene_noised))  # 新しい個体を
                          生成
157                     trial_P[i] = 0  # 試行回数をリセット
```

```
158
159                     logger("INFO", "Scout␣bee␣found␣a␣new␣food␣source")  # 新しい食料源を発見したことをログに記録
160                 end
161             end
162
163             if cvt_vorn_data_update <= cvt_vorn_data_update_limit  # ボロノイデータ更新回数が上限値以下の場合
164                 init_CVT(population)  # を初期化CVT
165
166                 new_archive = Archive(zeros(Int64, 0, 0), zeros(Int64, k_max), Dict{Int64, Individual}())  # 新しいアーカイブを生成
167                 archive     = deepcopy(cvt_mapping(population, new_archive))  # アーカイブを更新
168                 trial_A     = zeros(Int, k_max)  # 試行回数カウンタをリセット
169
170                 logger("INFO", "Recreate␣Voronoi␣diagram")  # ボロノイ図を再作成したことをログに記録
171             end
172         end
173
174         print(".")
175
176         return population, archive  # 更新された個体群とアーカイブを返す
177     end
178
179
180     # ABC algorithm
181     function ABC(population::Population, archive::Archive)
182         # Employee bee phase | 収穫蜂フェーズ
183         print("Employed␣bee␣phase␣")
184         population, archive = employed_bee(population, archive)
185         println(".␣Done")
186
187         # Onlooker bee phase | 追従蜂フェーズ
188         print("Onlooker␣bee␣phase␣")
189         population, archive = onlooker_bee(population, archive)
190         println(".␣Done")
191
192         # Scout bee phase | 偵察蜂フェーズ
193         print("Scout␣bee␣phase␣␣␣␣")
```

```
194         population, archive = scout_bee(population, archive)
195         println(". Done")
196 
197         return population, archive   # 更新された個体群とアーカイブを返す
198     end
```

参考文献

[1] D. J. VandenHeuvel: “DelaunayTriangulation.jl: A julia package for Delaunay triangulations and voronoi tessellations in the plane”, *Journal of Open Source Software*, Vol.9, No.101, p.7174 (2024-9)

[2] M. Besançon, T. Papamarkou, D. Anthoff, A. Arslan, S. Byrne, D. Lin, J. Pearson: “Distributions.jl: Definition and modeling of probability distributions in the juliastats ecosystem”, *Journal of Statistical Software*, Vol.98, No.16, pp.1–30 (2021)

[3] D. Lin, J. M. White, S. Byrne, D. Bates, A. Noack, J. Pearson, A. Arslan, K. Squire, D. Anthoff, T. Papamarkou, M. Besançon, J. Drugowitsch, M. Schauer, other contributors: “JuliaStats/Distributions.jl: a Julia package for probability distributions and associated functions” (2019-7)

[4] JuliaRandom: “StableRNGs.jl”, https://github.com/JuliaRandom/StableRNGs.jl (2024-12-10)

[5] JuliaIO: “FileIO.jl”, https://github.com/JuliaIO/FileIO.jl (2024-12-10)

[6] JuliaIO: “JLD2.jl”, https://github.com/JuliaIO/JLD2.jl (2024-12-10)

[7] S. Danisch, J. Krumbiegel: “Makie.jl: Flexible high-performance data visualization for Julia”, *Journal of Open Source Software*, Vol.6, No.65, p.3349 (2021)