R&D Europe (Germany)
Future Technology Research

MOO-EALib (Part 2): Functions

HRE-G / FTR Report 01 / 10

September 24, 2001

Tatsuya Okabe
Honda R&D Europe (Deutschland) GmbH
Future Technology Research
Carl-Legien Strasse 30
D-63073 Offenbach / Main
Germany

Correspondence:
tatsuya.okabe@hre-ftr.f.rd.honda.co.jp

Report No. TO-2001-04
Internal Report No. HRE-G/FTR 01/10

Contents

Chapter 1

Abstract

Most of optimization problems in the real world are Multiobjective Optimization (MOO) problems. Thus, these problems are very popular now and many researchers are developing several methods for MOO.

In order to try them, we needed the platform for MOO research. Thus, we developed *MOO-EALib* for multiobjective evolution strategies. This *MOO-EALib* is a C++ Class library. The original EALib was developed by Ruhr University Bochum in 1995. But, EALib was designed for Single Objective Optimization. Thus, we developed the extended version of EALib for MOO which we are calling *MOO-EALib*.

In the first report "MOO-EALib (Part 1): Structure", we explained the concept of *MOO-EALib* and the structure of *MOO-EALib* [1]. In this report, we explain the functions in *MOO-EALib*.

In Chapter 2, we explain *Class IndividualMOO* which corresponds to *Class Individual* in the original EALib. In Chapter 3, *Class PopulationMOO*, which corresponds to *Class Population*, is explained .

If you compare *MOO-EALib* with the original EALib, you can recoginize that many functions are the same. However, they were re-designed for MOO problems. Thus, we explained them in this report again.

Chapter 2

IndividualMOO

In this chapter, to clarify the explanation, the following words will be used:

Individual **:** individual which is for single objective optimization. This is generated by class *Individual*.

IndividualMOO **:** individual which is for multi objective optimization. This is generated by class *IndividualMOO*.

2.1 Internal variable

In class IndividualMOO, the following variables are defined.

MOOFitness : The vector to store the fitness variables. The type of this variable is *vector< double >*.

MOORank : The number of a rank. The type of this variable is *unsigned*.

MOOShare : The value of sharing. The type of this variable is *double*. Some times we can use this variable as a niche count.

2.2 Constructors

In the constructors, interval variables are initialized:

$$MOOFitness[i] = 0.0 \tag{2.1}$$
$$MOOFitness.size() = 0 \tag{2.2}$$
$$MOORank = 0 \tag{2.3}$$
$$MOOShare = 0.0 \tag{2.4}$$

MOOFitness.size( ) means the number of objective functions.

No. TO-IM-001

IndividualMOO( );

Generate an empty *IndividualMOO*.

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-IM-002

IndividualMOO(unsigned *n*);

Generate a new *IndividualMOO* and reserves memory for *n* chromosomes.

Parameters: *n* - The number of chromosomes.

Return Value: None.

Caveats: None.

No. TO-IM-003

IndividualMOO(unsigned *n*,
const Chromosome& *chrom*);

Generate an *IndividualMOO* that consists of *n* clones of the chromosome *chrom*.

Parameters: *n* - The number of chromosomes.
chrom - Chromosome to be cloned.

Return Value: None.

Caveats: None.

No. TO-IM-004

IndividuaMOO(const Chromosome& *chrom0*);

Generates an *IndividualMOO* that consists of the chromosome *chrom0*.

Parameters: *chrom0* - Chromosome to be cloned.

Return Value: None.

Caveats: None.

No. TO-IM-005

IndividualMOO(const Chromosome& *chrom0*,
const Chromosome& *chrom1*);

Generates an *IndividualMOO* that consists of the chromosomes *chrom0* and *chrom1*.

Parameters: *chrom0* - First chromosome, which is part of the new *IndividualMOO*.
chrom1 - Second chromosome, which is part of the new *IndividualMOO*.

Return Value: None.

Caveats: None.

No. TO-IM-006

IndividualMOO(const Chromosome& *chrom0 .. chrom2*);

Generates an *IndividualMOO* that consists of the chromosomes *chrom0* to *chrom2*.

Parameters: *chrom0 .. chrom2* - Chromosomes that make up the new *IndividualMOO*.

Return Value: None.

Caveats: None.

No. TO-IM-007

IndividualMOO(const Chromosome& *chrom0 .. chrom3*);

Generates an *IndividualMOO* that consists of the chromosomes *chrom0* to *chrom3*.

Parameters: *chrom0 .. chrom3* - Chromosomes, that make up the new *IndividualMOO*.

Return Value: None.

Caveats: None.

No. TO-IM-008

IndividualMOO(const Chromosome& *chrom0 .. chrom4*);

Generates an *IndividualMOO* that consists of the chromosomes *chrom0* to *chrom4*.

Parameters: *chrom0 .. chrom4* - Chromosomes, that make up the new *IndividualMOO*.

Return Value: None.

Caveats: None.

No. TO-IM-009

IndividualMOO(const Chromosome& *chrom0 .. chrom5*);

Generates an *IndividualMOO* that consist of the chromosomes *chrom0* to *chrom5*.

Parameters: *chrom0 .. chrom5* - Chromosomes, that make up the new *IndividualMOO*.

Return Value: None.

Caveats: None.

No. TO-IM-010

IndividualMOO(const Chromosome& *chrom0 .. chrom6*);

Generates an *IndividualMOO* that consist of the chromosomes *chrom0* to *chrom6*.

Parameters: *chrom0 .. chrom6* - Chromosomes, that make up the new *IndividualMOO*.

Return Value: None.

Caveats: None.

No. TO-IM-011

IndividualMOO(const Chromosome& *chrom0 .. chrom7*);

Generates an *IndividualMOO* that consist of the chromosomes *chrom0* to *chrom7*.

Parameters: *chrom0 .. chrom7* - Chromosomes, that make up the new *IndividualMOO*.

Return Value: None.

Caveats: None.

No. TO-IM-012

IndividualMOO(const vector< Chromosome* >& *chrom1*);

Generates an *IndividualMOO* that consists of the chromosomes stored in vector *chrom1*.

Parameters: *chrom1* - The vector of chromosomes.

Return Value: None.

Caveats: None.

No. TO-IM-013

IndividualMOO(const Individual& *indiv1*);

Generates an *IndividualMOO* that is a copy of *Individual indiv* including the internal class variables. Before copying, *Individual indiv* will be converted into *IndividualMOO*.

Parameters: *indiv1* - *Individual*.

Return Value: None.

Caveats: None.

No. TO-IM-014

IndividualMOO(const IndividualMOO& *indmoo*);

Generates an *IndividualMOO* that is a copy of *IndividualMOO indmoo* including the internal class variables.

Parameters: *indmoo* - *IndividualMOO*.

Return Value: None.

Caveats: None.

2.3 Destructor

No. TO-IM-015

~ **IndividualMOO**( );

Removes all chromosomes, that are contatined in the *IndividualMOO this* and then destroys *this* itself.

Parameters: None.

Return Value: None.

Caveats: None.

2.4 Operators

2.4.1 Assignment Operators

No. TO-IM-102

IndividualMOO& **operator** = (const IndividualMOO& *indmoo*);

Assigns the *IndividualMOO indmoo* (i.e. the chromosomes that are contained in *indmoo* and the values of the internal class variables) to *this*.

Parameters: *indmoo* - *IndividualMOO* that will be assigned to *this*.

Return Value: The *IndividualMOO this* with its new values.

Caveats: None.

No. TO-IM-104

IndividualMOO& **operator** = (const Individual& *ind*);

Assigns the *Individual ind* (i.e. the chromosomes that are contained in *ind* and the values of the internal class variables) to *this*. Before assigning, *Individual ind* will be converted into *IndividualMOO*.

Parameters: *ind* - *Individual* that will be assigned to *this*.

Return Value: The *IndividualMOO this* with its new values.

Caveats: None.

2.4.2 Comparison Operators

No. TO-IM-103

bool **operator** == (const IndividualMOO& *indmoo*) const;

Checks whether the current *IndividualMOO this* and the *IndividualMOO indmoo* are equal. The *IndividualMOO*s are equal, if they contain the same number of chromosomes, if any chromosome of *this* is equal to its corresponding chromosome in *ind* and if all values of the internal class variables are equal.

Parameters: *indmoo* - *IndividualMOO* that will be compared with *this*.

Return Value: *true* - The *IndividualMOO*s are equal,
false - The *IndividualMOO*s are not equal.

Caveats: None.

2.4.3 Operators for Extracting a Single Chromosome

No. TO-IM-100

Chromosome& **operator** [ ] (unsigned *i*);

Returns the chromosome with index *i* in the *IndividualMOO this*.

Parameters: *i* - Index of the chromosome in *this*, to be returned. *i* must be less than the number of chromosomes in *this*, otherwise the method will be aborted with an error message.

Return Value: Chromosome with index *i*.

Caveats: None.

No. TO-IM-101

const Chromosome& **operator** [ ] (unsigned *i*) const;

Returns the chromosome with index *i* in the *IndividualMOO this*.

Parameters: *i* - Index of the chromosome in *this*, to be returned. *i* must be less than the number of chromosomes in *this*, otherwise the method will be aborted with an error message.

Return Value: Chromosome with index *i*.

Caveats: None.

2.5 Information about the structure

No. TO-IM-020

unsigned **size**( ) const;

Returns the number of chromosome in *this*.

Parameters: None.

Return Value: Number of chromosomes in *this*.

Caveats: None.

No. TO-IM-021

unsigned **totalSize**( ) const;

Returns the number of all alleles of the chromosomes in *this*.

Parameters: None.

Return Value: The whole number of alleles in *this*.

Caveats: None.

2.6 Methods for internal variables

2.6.1 Internal variable *fitness*

No. TO-IM-030

void **setFitness**(double *fit*);

Sets the fitness and scaled fitness of *this* to the new value *fit*.

Parameters: *fit* - New value for the normal and scaled fitness.

Return Value: None.

Caveats: None.

No. TO-IM-031

double **fitnessValue**( ) const;

Returns the fitness value of the current *IndividualMOO this*.

Parameters: None.

Return Value: Fitness value of *this*.

Caveats: None.

No. TO-IM-032

double **getFitness**( ) const;

Returns the fitness value of the current *IndividualMOO this*.

Parameters: None.

Return Value: Fitness value of *this*.

Caveats: None.

2.6.2 Internal variable *scaledFitness*

No. TO-IM-033

void **setScaledFitness**(double *scalef*);

Sets the scaled fitness of *this* to the new value *scalef*.

Parameters: *scalef* - New value for the scaled fitness.

Return Value: None.

Caveats: None.

No. TO-IM-034

double **getScaledFitness**( ) const;

Returns the scaled fitness value of the current *IndividualMOO this*.

Parameters: None.

Return Value: Scaled fitness value of *this*.

Caveats: None.

2.6.3 Internal variable *age*

No. TO-IM-035

void **setAge**(unsigned *age*);

Sets the age of *this* to the new value *age*.

Parameters: *age* - New age of *this*. The default is 0.

Return Value: None.

Caveats: None.

No. TO-IM-036

void **incAge**( );

Increments the age of *this* by 1.

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-IM-037

unsigned **getAge**( ) const;

Returns the age of *this*, i.e. the generation.

Parameters: None.

Return Value: Age of *this*.

Caveats: None.

2.6.4 Internal variable *selProb*

No. TO-IM-038

void **setSelectionProbability**(double *prob*);

Sets the selection probability of *this* to the new value *prob*.

Parameters: *prob* - New selection probability for *this*.

Return Value: None.

Caveats: None.

No. TO-IM-039

void **setSelProb**(double *prob*);

Sets the selection probability of *this* to the new value *prob*.

Parameters: *prob* - New selection probability for *this*.

Return Value: None.

Caveats: None.

No. TO-IM-040

double **selectionProbability**( ) const;

Returns the selection probability of *this*.

Parameters: None.

Return Value: The selection probability of *this*.

Caveats: None.

No. TO-IM-041

double **getSelProb**( ) const;

Returns the selection probability of *this*.

Parameters: None.

Return Value: The selection probability of *this*.

Caveats: None.

2.6.5 Internal variable *numCopies*

No. TO-IM-042

void **setNumCopies**(unsigned *num*);

Sets the number of copies of *this* to the new value *num*.

Parameters: *num* - New value for the number of copies.

Return Value: None.

Caveats: None.

No. TO-IM-043

unsigned **numberOfCopies**( ) const;

Returns the number of reproductions of *this* that occured during the last selection.

Parameters: None.

Return Value: Number of reproductions of *this*.

Caveats: None.

No. TO-IM-044

unsigned **getNumCopies**( ) const;

Returns the number of reproductions of *this* that occured during the last selection.

Parameters: None.

Return Value: Number of reproductions of *this*.

Caveats: None.

2.6.6 Internal variable *evalFlg*

No. TO-IM-045

void **setEvaluationFlag**( );

Sets *evalFlg* to "true".

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-IM-046

void **clearEvaluationFlag**( );

Sets *evalFlg* to "false".

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-IM-047

void **setEvalFlg**(bool *flg*);

Sets the flag of evaluation of *this* to the new value *flg*.

Parameters: *flg* - New value for the flag of evaluation.

Return Value: None.

Caveats: None.

No. TO-IM-048

bool **needEvaluation**( ) const;

Returns the status of the *evalFlg*, i.e. whether the *IndividualMOO* must be evaluated.

Parameters: None.

Return Value: Status of *evalFlg*:
true - An evaluation of *this* is necessary,
false - The *IndividualMOO* needs no evaluation.

Caveats: None.

No. TO-IM-049

bool **getEvalFlg**( ) const;

Returns the status of the *evalFlg*, i.e. whether the *IndividualMOO* must be evaluated.

Parameters: None.

Return Value: Status of *evalFlg*:
true - An evaluation of *this* is necessary,
false - The *IndividualMOO* needs no evaluation.

Caveats: None.

2.6.7 Internal variable *feasible*

No. TO-IM-050

void **setFeasible**(bool *fea*);

Sets the *feasible* flag to the new value *fea*.

Parameters: *fea* - New value for the *feasible* flag:
true - The *IndividualMOO* is a possible solution for the current optimization problem,
false - The *IndividualMOO* does not represent a feasible solution.

Return Value: None.

Caveats: None.

No. TO-IM-051

bool **isFeasible**( ) const;

Returns whether *this* is a possible solution for the current optimization problem.

Parameters: None.

Return Value: Status of the *feasible*-flag:
true - The *IndividualMOO* is a possible solution,
false - The *IndividualMOO* cannot used as solution.

Caveats: None.

No. TO-IM-052

bool **getFeasible**( ) const;

Returns whether *this* is a possible solution for the current optimization problem.

Parameters: None.

Return Value: Status of the *feasible*-flag:
true - The *IndividualMOO* is a possible solution,
false - The *IndividualMOO* cannot used as solution.

Caveats: None.

2.6.8 Internal variable *elitist*

No. TO-IM-053

void **setElitist**(bool *eli*);

Sets the *elitist* flag to the new value *eli*.

Parameters: *eli* - New value for the *elitist* flag:
true - The *IndividualMOO* is a elitist,
false - The *IndividualMOO* is not a elitist.

Return Value: None.

Caveats: None.

No. TO-IM-054

bool **isElitist**( ) const;

Returns the status of the *elitist*-flag, i.e. whether *this* was chosen as elite *IndividualMOO* during the last selection.

Parameters: None.

Return Value: Status of the *elitist*-flag:
true - The *IndividualMOO* was chosen as elitist,
false - The *IndividualMOO* was not chosen.

Caveats: None.

No. TO-IM-055

bool **getElitist**( ) const;

Returns the status of the *elitist*-flag, i.e. whether *this* was chosen as elite *IndividualMOO* during the last selection.

Parameters: None.

Return Value: Status of the *elitist*-flag:
true - The *IndividualMOO* was chosen as elitist,
false - The *IndividualMOO* was not chosen.

Caveats: None.

2.7 Methods for internal variables (Individual-MOO)

2.7.1 The number of objective functions

No. TO-IM-060

void **setNoOfObj**(unsigned n);

Sets the number of objective functions to the new value n.

Parameters: n - New value for the number of objective functions.

Return Value: None.

Caveats: None.

No. TO-IM-061

unsigned **getNoOfObj**( );

Returns the number of objective functions.

Parameters: None.

Return Value: The number of objective functions.

Caveats: None.

2.7.2 Internal variable *MOORank*

No. TO-IM-062

void **setMOORank**(unsigned n);

Sets the rank to the new value n.

Parameters: n - New value for the rank.

Return Value: None.

Caveats: None.

No. TO-IM-063

unsigned **getMOORank**( );

Returns the rank of *this*.

Parameters: None.

Return Value: The rank of *this*.

Caveats: None.

2.7.3 Internal variable *MOOShare*

No. TO-IM-064

void **setMOOShare**(double *n*);

Sets the sharing value to the new value *n*.

Parameters: *n* - New value for the sharing (or the niche count).

Return Value: None.

Caveats: None.

No. TO-IM-065

unsigned **getMOOShare**( );

Returns the value of sharing (or the niche count) of *this*.

Parameters: None.

Return Value: The value of sharing (or the niche count) of *this*.

Caveats: None.

2.7.4 Internal variable *MOOFitness*

No. TO-IM-066

void **setMOOFitness**(unsigned *nof*,
double *fit*);

Sets the *nof* th fitness value to the new value *fit*.

Parameters: *nof* - The index of the fitness value which you want to change.
fit - New value for the fitness value.

Return Value: None.

Caveats: None.

No. TO-IM-067

double **getMOOFitness**(unsigned *nof*);

Gets the fitness value of the *nof* th objevtive function.

Parameters: *nof* - The index of the fitness value which you want to know.

Return Value: The fitness value of the *nof* th objective function.

Caveats: None.

No. TO-IM-068

void **setMOOFitnessValues**(double *f0*);

Sets the fitness value of the *0* th objevtive function to the new value *f0*.

Parameters: *f0* - New fitness value for the *0* th objective function.

Return Value: None.

Caveats: None.

No. TO-IM-069

void **setMOOFitnessValues**(double *f0*,
double *f1*);

Sets the fitness values of the objevtive functions to the new values *f0* .. *f1*.

Parameters: *f0* - New fitness value for the *0* th objective function.
f1 - New fitness value for the *1* st objective function.

Return Value: None.

Caveats: None.

No. TO-IM-070

void **setMOOFitnessValues**(double *f0* .. *f2*);

Sets the fitness values of the objevtive functions to the new values *f0* .. *f2*.

Parameters: *f0* .. *f2* - New fitness values for the objective functions.

Return Value: None.

Caveats: None.

No. TO-IM-071

void **setMOOFitnessValues**(double *f0* .. *f3*);

Sets the fitness values of the objevtive functions to the new values *f0* .. *f3*.

Parameters: *f0* .. *f3* - New fitness values for the objective functions.

Return Value: None.

Caveats: None.

No. TO-IM-072

void **setMOOFitnessValues**(double *f0 .. f4*);

Sets the fitness values of the objevtive functions to the new values *f0 .. f4*.

Parameters: *f0 .. f4* - New fitness values for the objective functions.

Return Value: None.

Caveats: None.

No. TO-IM-073

void **setMOOFitnessValues**(double *f0 .. f5*);

Sets the fitness values of the objevtive functions to the new values *f0 .. f5*.

Parameters: *f0 .. f5* - New fitness values for the objective functions.

Return Value: None.

Caveats: None.

No. TO-IM-074

void **setMOOFitnessValues**(double *f0 .. f6*);

Sets the fitness values of the objevtive functions to the new values *f0 .. f6*.

Parameters: *f0 .. f6* - New fitness values for the objective functions.

Return Value: None.

Caveats: None.

No. TO-IM-075

void **setMOOFitnessValues**(double *f0 .. f7*);

Sets the fitness values of the objevtive functions to the new values *f0 .. f7*.

Parameters: *f0 .. f7* - New fitness values for the objective functions.

Return Value: None.

Caveats: None.

No. TO-IM-076

void **setMOOFitnessValues**(vector< double >& *fit*);

Sets the fitness values of the objevtive functions to the new values in the vector *fit*.

Parameters: *fit* - The vector which stores new fitness values for the objective functions.

Return Value: None.

Caveats: None.

No. TO-IM-077

vector< double >& **getMOOFitnessValues**( );

Returns the fitness values.

Parameters: None.

Return Value: The fitness values.

Caveats: None.

No. TO-IM-078

void **initializeMOOFitness**(double x);

Sets the fitness values of all objevtive functions to the new value x.

Parameters: x - New value for all fitness values.

Return Value: None.

Caveats: None.

2.8 Change the structure

No. TO-IM-110

void **replace**(unsigned i,
const Chromosome& *chrom*);

Replaces chromosome number i with the content of chromosome *chrom*.

Parameters: i - Index of the chromosome of *this*, to be replaced. i must be less than the number of chromosomes in *this*, otherwise the method will be aborted with an error message.

chrom - Chromosome that replaces the old chromosome in *this*.

Return Value: None.

Caveats: None.

No. TO-IM-111

void **insert**(unsigned *i*,
const Chromosome& *chrom*);

Inserts chromosome *chrom* at position *i* into *this*.

Parameters: *i* - Insertion-position in *this*. The maximum value for *i* shall be equal to the number of chromosomes in *this*, otherwise the method will be aborted with an error message.

chrom - Chromosome to be inserted into *this*.

Return Value: None.

Caveats: None.

No. TO-IM-112

void **append**(const Chromosome& *chrom*);

Appends the chromosome *chrom* at the end of *this*.

Parameters: *chrom* - Chromosome to be appended at the end of *this*.

Return Value: None.

Caveats: None.

No. TO-IM-113

void **remove**(unsigned *i*);

Removes the chromosome with index *i* from *this*.

Parameters: *i* - Index of the chromosome to be removed from *this*. *i* must be less than the number of chromosomes in *this*, otherwise the method will be aborted with an error message.

Return Value: None.

Caveats: None.

No. TO-IM-114

void **remove**(unsigned *i*,
unsigned *j*);

Removes all chromosomes with indices in the range $[i, j]$.

Parameters: *i* - Index of the first chromosome to be removed. *i* must be less than the number of chromosomes in *this* and not greater as *j* or the method will be aborted with an error message.

j - Index of the last chromosome to be removed. *j* must be less than the number of chromosomes in *this* and not smaller as *i* or the method will be aborted with an error message.

Return Value: None.

Caveats: None.

2.9 Aggregation Methods

No. TO-IM-200

double **aggregation**(const vector< double >& *weight*);

Calculate the weighted sum of fitness values: $\sum_{i=0}^{n-1} w_i f_i$.

Parameters: *weight* - The vector which stores the values of weight for an aggregation method.

Return Value: The weighted sum of fitness values.

Caveats: None.

No. TO-IM-201

double **simplesum**( ) const;

Calculate the sum of fitness values: $\sum_{i=0}^{n-1} f_i$.

Parameters: None.

Return Value: The sum of fitness values.

Caveats: None.

2.10 Output data

No. TO-IM-500

---

void **printIM**( ) const;

Outputs the data of the individual *this*.

Parameters: None.

Return Value: None.

Caveats: None.

Chapter 3

PopulationMOO

In this chapter, to clarify the explanation, the following words will be used:

***Individual* :** individual which is for single objective optimization. This is generated by Class *Individual*.

***IndividualMOO* :** individual which is for multi objective optimization. This is generated by Class *IndividualMOO*.

***Population* :** population which is for single objective optimization and stores *Individuals*. This is generated by Class *Population*.

***PopulationMOO* :** population which is for multi objective optimization and stores *IndividualMOOs*. This is generated by Class *PopulationMOO*.

3.1 Constructors

No. TO-PM-001

PopulationMOO( );

Generates a new *PopulationMOO*.

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-PM-002

PopulationMOO(unsigned n);

Generates a new *PopulationMOO* and reserves space for n *IndividualMOOs* of undefined type.

Parameters: n - Number of *IndividualMOOs* in the new *PopulationMOO*.

Return Value: None.

Caveats: None.

No. TO-PM-003

PopulationMOO(const IndividualMOO& *indmoo*);

Generates a new *PopulationMOO* that consists of the *IndividualMOO indmoo*.

Parameters: *indmoo* - *IndividualMOO* that makes up the new *PopulationMOO*.

Return Value: None.

Caveats: None.

No. TO-PM-004

PopulationMOO(unsigned *n*,
const IndividualMOO& *indmoo*);

Generates a new *PopulationMOO* consisting of *n* copies of the *IndividualMOO indmoo*.

Parameters: *n* - Number of copies of the *IndividualMOO indmoo* to form the new *PopulationMOO*.
indmoo - *IndividualMOO* that makes up the new *PopulationMOO*.

Return Value: None.

Caveats: None.

No. TO-PM-005

PopulationMOO(unsigned *n*,
const Chromosome& *chrom0*);

Generates a new *PopulationMOO* consisting of *n IndividualMOO*s that are formed by cloning chromosome *chrom0*.

Parameters: *n* - Number of *IndividualMOO*s that make up the new *PopulationMOO*.
chrom0 - Chromosome.

Return Value: None.

Caveats: None.

No. TO-PM-006

PopulationMOO(unsigned *n*,
const Chromosome& *chrom0 .. chrom1*);

Generates a new *PopulationMOO* that consists of *n IndividualMOO*s formed by cloning the chromosomes *chrom0* and *chrom1*.

Parameters: *n* - Number of *IndividualMOO*s that make up the new *PopulationMOO*.
chrom0 .. chrom1 - Chromosomes to be cloned.

Return Value: None.

Caveats: None.

No. TO-PM-007

PopulationMOO(unsigned *n*,
const Chromosome& *chrom0 .. chrom2*);

Generates a new *PopulationMOO* that consists of *n IndividualMOO*s formed by cloning the chromosomes *chrom0* to *chrom2*.

Parameters: *n* - Number of *IndividualMOO*s that make up the new *PopulationMOO*.
chrom0 .. chrom2 - Chromosomes to be cloned.

Return Value: None.

Caveats: None.

No. TO-PM-008

PopulationMOO(unsigned *n*,
const Chromosome& *chrom0 .. chrom3*);

Generates a new *PopulationMOO* that consists of *n IndividualMOO*s formed by cloning the chromosomes *chrom0* to *chrom3*.

Parameters: *n* - Number of *IndividualMOO*s that make up the new *PopulationMOO*.
chrom0 .. chrom3 - Chromosomes to be cloned.

Return Value: None.

Caveats: None.

No. TO-PM-009

PopulationMOO(unsigned *n*,
const Chromosome& *chrom0 .. chrom4*);

Generates a new *PopulationMOO* that consists of *n IndividualMOO*s formed by cloning the chromosomes *chrom0* to *chrom4*.

Parameters: *n* - Number of *IndividualMOO*s that make up the new *PopulationMOO*.
chrom0 .. chrom4 - Chromosomes to be cloned.

Return Value: None.

Caveats: None.

No. TO-PM-010

PopulationMOO(unsigned *n*,
const Chromosome& *chrom0 .. chrom5*);

Generates a new *PopulationMOO* that consists of *n IndividualMOO*s formed by cloning the chromosomes *chrom0* to *chrom5*.

Parameters: *n* - Number of *IndividualMOO*s that make up the new *PopulationMOO*.
chrom0 .. chrom5 - Chromosomes to be cloned.

Return Value: None.

Caveats: None.

No. TO-PM-011

PopulationMOO(unsigned *n*,
const Chromosome& *chrom0 .. chrom6*);

Generates a new *PopulationMOO* that consists of *n IndividualMOO*s formed by cloning the chromosomes *chrom0* to *chrom6*.

Parameters: *n* - Number of *IndividualMOO*s that make up the new *PopulationMOO*.
chrom0 .. chrom6 - Chromosomes to be cloned.

Return Value: None.

Caveats: None.

No. TO-PM-012

PopulationMOO(unsigned *n*,
const Chromosome& *chrom0 .. chrom7*);

Generates a new *PopulationMOO* that consists of *n IndividualMOO*s formed by cloning the chromosomes *chrom0* to *chrom7*.

Parameters: *n* - Number of *IndividualMOO*s that make up the new *PopulationMOO*.
chrom0 .. chrom7 - Chromosomes to be cloned.

Return Value: None.

Caveats: None.

No. TO-PM-013

PopulationMOO(unsigned *n*,
const vector< Chromosome * >& *chrom*);

Generates a new *PopulationMOO* that consists of *n IndividualMOO*s formed by cloning the chromosomes that are stored in vector *chrom*.

Parameters: *n* - Number of *IndividualMOO*s that make up the new *PopulationMOO*.
chrom - Vector with chromosomes to be cloned.

Return Value: None.

Caveats: None.

No. TO-PM-014

PopulationMOO(const PopulationMOO& *popmoo*);

Generates a new *PopulationMOO* that consists of a copy of the *PopulationMOO popmoo* including the values for the flags *ascending*, *spinOnce*, *subPop* and *index*.

Parameters: *popmoo* - *PopulationMOO* whose copy will form the new *PopulationMOO*.

Return Value: None.

Caveats: None.

No. TO-PM-015

PopulationMOO(const Population& *pop*);

Generates a new *PopulationMOO* that consists of a copy of the *Population pop* including the values for the flags *ascending*, *spinOnce*, *subPop* and *index*. Before generating a new *PopulationMOO*, *Population pop* will be converted into *PopulationMOO*.

Parameters: *pop* - *Population* whose copy will form the new *PopulationMOO*.

Return Value: None.

Caveats: None.

3.2 Destructor

No. TO-PM-016

~PopulationMOO( );

If the destructor is called by a subpopulation nothing happens, else all *IndividualMOO*s in the *PopulationMOO* are removed and the *PopulationMOO* itself is destroyed.

Parameters: None.

Return Value: None.

Caveats: The destructor is virtual. The destructor should not be called directly.

3.3 Operators

3.3.1 Assignment Operators

No. TO-PM-042

PopulationMOO& **operator** = (const IndividualMOO& *indmoo*);

The values of the *IndividualMOO indmoo* will be assigned to all *IndividualMOO*s of *this*.

Parameters: *indmoo* - *IndividualMOO*.

Return Value: The *PopulationMOO* with its new *IndividualMOO*s.

Caveats: None.

No. TO-PM-043

PopulationMOO& **operator** = (const Individual& *ind*);

The values of the *Individual ind* will be assigned to all *IndividualMOO*s of *this*. Before assignment, *Individual ind* will be converted into *IndividualMOO*.

Parameters: *ind* - *Individual*.

Return Value: The *PopulationMOO* with its new *IndividualMOO*s.

Caveats: None.

No. TO-PM-044

PopulationMOO& **operator** = (const PopulationMOO& *popmoo*);

Assigns all *IndividualMOO*s of *popmoo* to *this*. This method will only work if both *PopulationMOO*s consist of the same number of *IndividualMOO*s or if *this* is no subpopulation, otherwise the method will be aborted with an error message.

Parameters: *popmoo* - *PopulationMOO*.

Return Value: The *PopulationMOO* with its new *IndividualMOO*s.

Caveats: None.

No. TO-PM-045

PopulationMOO& **operator** = (const Population& *pop*);

Assigns all *Individual*s of *pop* to *this*. Before assignment, all *Individual*s in *Population pop* will be converted into *IndividualMOO*s. This method will only work if both *PopulationMOO* and *Population* consist of the same number of individuals (*IndividualMOO* or *Individual*) or if *this* is no subpopulation, otherwise the method will be aborted with an error message.

Parameters: *pop* - *Population*.

Return Value: The *PopulationMOO* with its new *IndividualMOO*s.

Caveats: None.

3.3.2 Comparison Operators

No. TO-PM-046

---

bool **operator** = =(const PopulationMOO& *popmoo*) const;

Tests whether the *PopulationMOO this* and *popmoo* are equal. *this* and *popmoo* are equal, if they contain the same number of *IndividualMOO*s, all *IndividualMOO*s of *this* are equal to the corresponding *IndividualMOO*s of *popmoo* and the values of all internal variables of *this* and *popmoo* are the same.

Parameters: *popmoo* - *PopulationMOO* that shall be compared with *this*.

Return Value: *true* - *this* and *popmoo* are equal,
false - *this* and *popmoo* are different.

Caveats: None.

3.3.3 Operators for Extracting a Single IndividualMOO

No. TO-PM-040

IndividualMOO& **operator** [ ](unsigned *i*);

Returns *IndividualMOO* number *i* out of *this*.

Parameters: *i* - Index of the *IndividualMOO* to be returned. *i* must be less than the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.

Return Value: The *IndividualMOO* numbered *i*.

Caveats: None.

No. TO-PM-041

---

const IndividualMOO& **operator** [ ](unsigned *i*) const;

Returns *IndividualMOO* number *i* out of *this*.

Parameters: *i* - Index of the *IndividualMOO* to be returned. *i* must be less than the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.

Return Value: The *IndividualMOO* numbered *i*.

Caveats: None.

3.4 Information about Class *PopulationMOO*

No. TO-PM-030

unsigned **size**( ) const;

Returns the number of IndividualMOOs in *this*.

Parameters: None.

Return Value: Number of *IndividualMOO*s in *this*.

Caveats: None.

No. TO-PM-031

Individual** **begin**( );

Returns the first data of the vector in *this*.

Parameters: None.

Return Value: The first data of the vector in *this*.

Caveats: None.

No. TO-PM-032

Individual** **end**( );

Returns the last data of the vector in *this*.

Parameters: None.

Return Value: The last data of the vector in *this*.

Caveats: None.

3.5 Methods for internal variables

3.5.1 Internal variable *ascending*

No. TO-PM-050

void **setMaximize**( );

Sets the value of the *ascending*-flag to "false", so all *IndividualMOO*s will be sorted by descending fitness values *fitness*.

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-PM-051

void **setMinimize**( );

Sets the value of the *ascending*-flag to "true", so all *IndividualMOO*s will be sorted by ascending fitness values *fitness*.

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-PM-052

void **setAscending**(bool *strategy*);

Sets the *ascending*-flag to the new value *strategy*.

Parameters: *strategy* - New value for the *ascending*-flag.
true - Minimize. The *IndividualMOO*s are sorted by ascending fitness values *fitness*.
false - Maximize. The *IndividualMOO*s are sorted by descending fitness values *fitness*.

Return Value: None.

Caveats: None.

No. TO-PM-053

bool **ascendingFitness**( ) const;

Returns whether the *IndividualMOO*s inside *this* are sorted by ascending or descending fitness values *fitness*.

Parameters: None.

Return Value: Value of the *ascending*-flag:
true - The *IndividualMOO*s are sorted by ascending fitness values *fitness*,
false - The *IndividualMOO*s are sorted by descending fitness values *fitness*.

Caveats: None.

No. TO-PM-054

bool **getAscending**( ) const;

Returns whether the *IndividualMOO*s inside *this* are sorted by ascending or descending fitness values *fitness*.

Parameters: None.

Return Value: Value of the *ascending*-flag:
true - The *IndividualMOO*s are sorted by ascending fitness values *fitness*,
false - The *IndividualMOO*s are sorted by descending fitness values *fitness*.

Caveats: None.

3.5.2 Internal variable *spinOnce*

No. TO-PM-055

void **spinWheelOneTime**( );

Sets the value of the *spinOnce*-flag to "true", so the roulette wheel will be spinned only one time during the next *selectRouletteWheel*-method call.

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-PM-056

void **spinWheelMultipleTimes**( );

Sets the value of the *spinOnce*-flag to "false", so the roulette wheel will be spinned several times during the next *selectRouletteWheel*-method call.

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-PM-057

void **setSpinOnce**(bool *spin*);

Sets *spinOnce*-flag to the new value *spin*.

Parameters: *spin* - New value for *spinOnce*-flag.
true - the roulette wheel will be spinned only one time.
false - the roulette wheel will be spinned several times.

Return Value: None.

Caveats: None.

No. TO-PM-058

bool **getSpinOnce**( ) const;

Returns *spinOnce*-flag.
true - the roulette wheel will be spinned only one time.
false - the roulette wheel will be spinned several times.

Parameters: None.

Return Value: The value of *spinOnce*-flag.

Caveats: None.

3.5.3 Internal variable *index*

No. TO-PM-059

void **setIndex**(unsigned *i*);

Sets *index* to the new value *i*.

Parameters: *i* - New value for *index* of *this*.

Return Value: None.

Caveats: None.

No. TO-PM-060

unsigned **getIndex**( ) const;

Returns the value of *index*.

Parameters: None.

Return Value: The value of *index*.

Caveats: None.

3.5.4 Internal variable *subPop*

No. TO-PM-061

void **setSubPop**(bool *sub*);

Sets *subPop* to the new value *sub*.

Parameters: *sub* - New value for *subPop* of *this*.
true - This *PopulationMOO* is a sub *PopulationMOO*.
false - This *PopulationMOO* is not a sub *PopulationMOO*.

Return Value: None.

Caveats: None.

No. TO-PM-062

bool **getSubPop**( ) const;

Returns the value of *subPop*.
true - This *PopulationMOO* is a sub *PopulationMOO*.
false - This *PopulationMOO* is not a sub *PopulationMOO*.

Parameters: None.

Return Value: The value of *subPop*.

Caveats: None.

3.5.5 Internal variable *age* in the class *Individual*

No. TO-PM-124

void **setAge**(unsigned *a*);

Sets the age of all *IndividualMOO*s in *this* to the new value *a*.

Parameters: *a* - New age of the *IndividualMOO*s. The default value is 0.

Return Value: None.

Caveats: None.

No. TO-PM-125

void **incAge**( );

Increases the age of all *IndividualMOO*s inside *this* by 1.

Parameters: None.

Return Value: None.

Caveats: None.

3.6 Methods for internal variables (IndividualMOO)

3.6.1 The number of objective functions

No. TO-IM-200

void **setNoOfObj**(unsigned *NOO*);

Sets the number of objective functions inside all *IndividualMOO*s to the new value *NOO*.

Parameters: *NOO* - New value for the number of objective functions.

Return Value: None.

Caveats: None.

3.6.2 Internal variable *MOOFitness*

No. TO-IM-201

void **setMOOFitness**(double *obj*);

Sets all *MOOFitness* in *this* to the new value *obj*.

Parameters: *obj* - New value for the fitness values of all objective functions inside all *IndividualMOO*s.

Return Value: None.

Caveats: None.

3.6.3 Internal variable *MOORank*

No. TO-IM-202

void **setMOORank**(unsigned *MOOR*);

Sets all *MOORank* in all *IndividualMOO*s to the new value *MOOR*.

Parameters: *MOOR* - New value for *MOORank* of all *IndividualMOO*s.

Return Value: None.

Caveats: None.

3.6.4 Internal variable *MOOShare*

No. TO-IM-203

---

void **setMOOShare**(double *MOOS*);

Sets all *MOOShare* in all *IndividualMOO*s to the new value *MOOS*.

Parameters: *MOOS* - New value for *MOOShare* of all *IndividualMOO*s.

Return Value: None.

Caveats: None.

3.7 Change the structure

3.7.1 The number of *IndividualMOO*s

No. TO-PM-033

---

void **resize**(unsigned *n*);

Changes the size of the *PopulationMOO* to the new value *n*. If *n* is less than the old size *m* of *this*, then all *m* - *n* *IndividualMOO*s at the end of *this* will be removed, if *n* is greater than *m*, then *n* - *m* new empty *IndividualMOO*s will be appended at the end of *this*. All other *IndividualMOO*s will remain unchanged.

Parameters: *n* - New size of the *PopulationMOO*.

Return Value: None.

Caveats: None.

3.7.2 Replace *IndividualMOO*s

No. TO-PM-070

---

void **replace**(unsigned *i*,
const Individual& *ind*);

Replaces *IndividualMOO* number *i* by the *Individual ind*.

Parameters: *i* - Index of the *IndividualMOO* inside of *this* to be replaced. *i* must be less than the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.

ind - *Individual* to replace the old one. First, *Individual ind* will be converted into an *IndividualMOO*.

Return Value: None.

Caveats: None.

No. TO-PM-071

void **replace**(unsigned *i*,
const IndividualMOO& *indmoo*);

Replaces *IndividualMOO* number *i* by the *IndividualMOO indmoo*.

Parameters: *i* - Index of the *IndividualMOO* inside of *this* to be replaced. *i* must be less than the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.
indmoo - *IndividualMOO* to replace the old one.

Return Value: None.

Caveats: None.

No. TO-PM-072

void **replace**(unsigned *i*,
const Population& *pop*);

Replaces all *IndividualMOO*s inside of *this*, begining with the one at position number *i*, by the *IndividualMOO*s which are converted from *Individual*s of the *Population pop*. If the number of *Individual*s inside of *pop* is greater than the number of *IndividualMOO*s in *this* begining at position *i*, then the method will be aborted with an error message.

Parameters: *i* - Index of the first *IndividualMOO* inside of *this* to be replaced.
pop - *Population* to replace the old *IndividualMOO*s.

Return Value: None.

Caveats: None.

No. TO-PM-073

void **replace**(unsigned *i*,
const PopulationMOO& *popmoo*);

Replaces all *IndividualMOO*s inside of *this*, begining with the one at position number *i*, by the *IndividualMOO*s in the *PopulationMOO popmoo*. If the number of *IndividualMOO*s inside of *popmoo* is greater than the number of *IndividualMOO*s in *this* begining at position *i*, then the method will be aborted with an error message.

Parameters: *i* - Index of the first *IndividualMOO* inside of *this* to be replaced.
popmoo - *PopulationMOO* to replace the old *IndividualMOO*s.

Return Value: None.

Caveats: None.

3.7.3 Insert *IndividualMOO*s

No. TO-PM-074

---

void **insert**(unsigned *i*,
const Individual& *ind*);

Inserts the *Individual ind* at index *i* inside the *PopulationMOO this*. Before insertion, *Individual ind* will be converted into *IndividualMOO*.

Parameters: *i* - Index at which *ind* is inserted in *this*. *i* must be at most as great as the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.
ind - *Individual* to be inserted.

Return Value: None.

Caveats: None.

No. TO-PM-075

---

void **insert**(unsigned *i*,
const IndividualMOO& *indmoo*);

Inserts the *IndividualMOO indmoo* at index *i* inside the *PopulationMOO this*.

Parameters: *i* - Index at which *indmoo* is inserted in *this*. *i* must be at most as great as the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.
indmoo - *IndividualMOO* to be inserted.

Return Value: None.

Caveats: None.

No. TO-PM-076

---

void **insert**(unsigned *i*,
const Population& *pop*);

Inserts all *Individual*s of *pop* into *this*, beginning at position *i*. Before insertion, all *Individual*s will be converted into *IndividualMOO*s.

Parameters: *i* - Index of *this* where the first *Individual* of *pop* is inserted. *i* must be at most as great as the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.
pop - *Population* with the *Individual*s that will be inserted into *this*.

Return Value: None.

Caveats: None.

No. TO-PM-077

void **insert**(unsigned *i*,
const PopulationMOO& *popmoo*);

Inserts all *IndividualMOO*s of *popmoo* into *this*, beginning at position *i*.

Parameters: *i* - Index of *this* where the first *IndividualMOO* of *popmoo* is inserted. *i* must be at most as great as the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.

popmoo - *PopulationMOO* with the *IndividualMOO*s that will be inserted into *this*.

Return Value: None.

Caveats: None.

3.7.4 Append *IndividualMOO*s

No. TO-PM-078

void **append**(const Individual& *ind*);

Appends *Individual ind* at the end of *this*. Before appending, *Individual ind* will be converted into *IndividualMOO*.

Parameters: *ind* - The *Individual* to be appended to *this*.

Return Value: None.

Caveats: None.

No. TO-PM-079

void **append**(const IndividualMOO& *indmoo*);

Appends *IndividualMOO indmoo* at the end of *this*.

Parameters: *indmoo* - The *IndividualMOO* to be appended to *this*.

Return Value: None.

Caveats: None.

No. TO-PM-080

void **append**(const Population& *pop*);

Appends all *Individuals* of *Population pop* at the end of *PopulationMOO this*. Before appending, all *Individuals* of *Population pop* will be converted into *IndividualMOO*s.

Parameters: *pop* - *Population* to be appended to *this*.

Return Value: None.

Caveats: None.

No. TO-PM-081

void **append**(const PopulationMOO& *popmoo*);

Appends all *IndividualMOO*s of *PopulationMOO popmoo* at the end of *PopulationMOO this*.

Parameters: *popmoo* - *PopulationMOO* to be appended to *this*.

Return Value: None.

Caveats: None.

3.7.5 Remove *IndividualMOO*s

No. TO-PM-082

void **remove**(unsigned *i*);

Removes *IndividualMOO* number *i* from *this*.

Parameters: *i* - Index of the *IndividualMOO* that is to be removed from *this*. *i* must be less than the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.

Return Value: None.

Caveats: None.

No. TO-PM-083

void **remove**(unsigned *i*,
unsigned *k*);

Removes all *IndividualMOO*s in the range $[i, k]$ from the *PopulationMOO this*.

Parameters: *i* - Index of the first *IndividualMOO* of *this* to be removed. *i* must be less than the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message. Additionally, *i* must be at most as great as *k* or no *IndividualMOO* will be removed from *this*.

k - Index of the last *IndividualMOO* to be removed from *this*. *k* must be less than the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.

Return Value: None.

Caveats: None.

3.7.6 Exchange all *IndividualMOO*s

No. TO-PM-098

void **exchange**(PopulationMOO& *popmoo*);

Swaps all *IndividualMOO*s of *popmoo* with the *IndividualMOO*s of *this*. The *PopulationMOO*s must contain the same number of *IndividualMOO*s, otherwise the method will be aborted with an error message.

Parameters: *popmoo* - *PopulationMOO* to be exchanged.

Return Value: None.

Caveats: None.

3.8 Change the order of *IndividualMOO*s

No. TO-PM-090

void **swap**(unsigned *i*,
unsigned *j*);

Swaps *IndividualMOO*s number *i* and *j*. The method also works properly, when *i* is greater than *j*.

Parameters: *i* - Index of the first *IndividualMOO*. *i* must be less than the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.

j - Index of the second *IndividualMOO*. *j* must be less than the number of *IndividualMOO*s in *this*, otherwise the method will be aborted with an error message.

Return Value: None.

Caveats: None.

No. TO-PM-093

void **sortIndividuals**(vector< Individual * >& *indvec*);

Sorts all *Individual*s or *IndividualMOO*s in *indvec* according to *ascending*-flag. If the flag is set to "true", then all *Individual*s or *IndividualMOO*s are sorted by ascending fitness *fitness*, otherwise by descending fitness *fitness*.

Parameters: *indvec* - Vector with *Individual*s or *IndividualMOO*s to be sorted.

Return Value: None.

Caveats: None.

No. TO-PM-094

void **sort**( );

Sorts all *Individual*s or *IndividualMOO*s inside of *this* according to the setting of the *ascending*-flag. If the flag is set to "true", then the *Individual*s or *IndividualMOO*s will be sorted by ascending fitness values *fitness*, if it is set to "false", they will be sorted by descending fitness values *fitness*.

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-PM-095

void **shuffle**( );

Randomly changes the positions of the *Individual*s or *IndividualMOO*s inside of *this*.

Parameters: None.

Return Value: None.

Caveats: None.

3.9 Compare *IndividualMOO*s

No. TO-PM-091

bool **lessFitness**(IndividualMOO*const& *i1*,
IndividualMOO*const& *i2*);

Returns whether the fitness of *IndividualMOO i1* is less than the fitness of *IndividualMOO i2*.

Parameters: *i1* - The first *IndividualMOO* to be compared.
i2 - The second *IndividualMOO* to be compared.

Return Value: Status of the fitness comparison:
true - The fitness of *IndividualMOO i1* is less than the fitness of *IndividualMOO i2*,
false - The fitness of *IndividualMOO i1* is greater than or equal to the fitness of *IndividualMOO i2*.

Caveats: None.

No. TO-PM-092

bool **greaterFitness**(IndividualMOO*const& *i1*,
IndividualMOO*const& *i2*);

Returns whether the fitness of *IndividualMOO i1* is greater than the fitness of *IndividualMOO i2*.

Parameters: *i1* - The first *IndividualMOO* to be compared.
i2 - The second *IndividualMOO* to be compared.

Return Value: Status of the fitness comparison:
true - The fitness of *IndividualMOO i1* is greater than the fitness of *IndividualMOO i2*,
false - The fitness of *IndividualMOO i1* is less than or equal to the fitness of *IndividualMOO i2*.

Caveats: None.

No. TO-PM-096

bool **greaterScoreAscending**(IndividualMOO* const& *i1*,
IndividualMOO* const& *i2*);

Auxiliary method for *selectEPTournament*. Checks whether the scaled fitness of *IndividualMOO i1* is greater or whether the normal fitness is less than the corresponding values of *IndividualMOO i2*.

Parameters: *i1* - The first *IndividualMOO* to be compared.
i2 - The second *IndividualMOO* to be compared.

Return Value: *true* - The scaled fitness of *IndividualMOO i1* is greater than the scaled fitness of *IndividualMOO i2* or the normal fitness of *IndividualMOO i1* is less than the normal fitness of *IndividualMOO i2*.
false - The scaled fitness of *IndividualMOO i1* is less than the scaled fitness of *IndividualMOO i2* or the normal fitness of *IndividualMOO i1* is greater than the normal fitness of *IndividualMOO i2*.

Caveats: None.

No. TO-PM-097

bool **greaterScoreDescending**(IndividualMOO*const& *i1*,
IndividualMOO*const& *i2*);

Auxiliary method for *selectEPTournament*. Checks whether the scaled or normal fitness of *IndividualMOO i1* is greater than the corresponding values of *IndividualMOO i2*.

Parameters: *i1* - The first *IndividualMOO* to be compared.
i2 - The second *IndividualMOO* to be compared.

Return Value: *true* - The scaled fitness of *IndividualMOO i1* is less than the scaled fitness of *IndividualMOO i2* or the normal fitness of *IndividualMOO i1* is greater than the normal fitness of *IndividualMOO i2*.
false - The scaled fitness of *IndividualMOO i1* is greater than the scaled fitness of *IndividualMOO i2* or the normal fitness of *IndividualMOO i1* is less than the normal fitness of *IndividualMOO i2*.

Caveats: None.

No. TO-PM-110

IndividualMOO& **best**(IndividualMOO& *i1*,
IndividualMOO& *i2*) const;

Returns the *IndividualMOO* with the better fitness values.

Parameters: *i1* - The first *IndividualMOO* to be compared.
i2 - The second *IndividualMOO* to be compared.

Return Value: The *IndividualMOO* with the "better" fitness according to *ascending*-flag. If the flag sets to "true", then the *IndividualMOO* with the smaller fitness value will be returned, if the flag sets to "false", the *IndividualMOO* with the greater fitness value is returned.

Caveats: None.

No. TO-PM-111

IndividualMOO& **best**(unsigned *i*,
unsigned *j*) const;

Returns the *IndividualMOO* with the better fitness values.

Parameters: *i* - The index of the first *IndividualMOO* to be compared.

j - The index of the second *IndividualMOO* to be compared.

Return Value: The *IndividualMOO* with the "better" fitness according to *ascending*-flag. If the flag sets to "true", then the *IndividualMOO* with the smaller fitness value will be returned, if the flag sets to "false", the *IndividualMOO* with the greater fitness value is returned.

Caveats: None.

No. TO-PM-112

IndividualMOO& **worst**(IndividualMOO& *i1*,
IndividualMOO& *i2*) const;

Returns the *IndividualMOO* with the worst fitness values.

Parameters: *i1* - The first *IndividualMOO* to be compared.

i2 - The second *IndividualMOO* to be compared.

Return Value: The *IndividualMOO* with the "worst" fitness value according to *ascending*-flag. If the flag sets to "true", then the *IndividualMOO* with the greater fitness value will be returned, if the flag sets to "false", the *IndividualMOO* with the smaller fitness value is returned.

Caveats: None.

No. TO-PM-113

IndividualMOO& **worst**(unsigned *i*,
unsigned *j*);

Returns the *IndividualMOO* with the worst fitness values.

Parameters: *i* - The index of the first *IndividualMOO* to be compared.

j - The index of the second *IndividualMOO* to be compared.

Return Value: The *IndividualMOO* with the "worst" fitness according to *ascending*-flag. If the flag sets to "true", then the *IndividualMOO* with the greater fitness value will be returned, if the flag sets to "false", the *IndividualMOO* with the smaller fitness value is returned.

Caveats: None.

3.10 Information about *IndividualMOO*s

No. TO-PM-120

double **minFitness**( ) const;

Returns the minimum fitness value inside *this*. This value is selected from all fitness values *fitness*, not MOO fitness values *MOOFitness*.

Parameters: None.

Return Value: Minimum fitness value of the *PopulationMOO* or 0.0, if the *PopulationMOO* is empty.

Caveats: None.

No. TO-PM-121

double **maxFitness**( ) const;

Returns the maximum fitness value inside *this*. This value is selected from all fitness values *fitness*, not MOO fitness values *MOOFitness*.

Parameters: None.

Return Value: Maximum fitness value of the *PopulationMOO* or 0.0, if the *PopulationMOO* is empty.

Caveats: None.

No. TO-PM-122

double **meanFitness**( ) const;

Returns the mean fitness value of all *IndividualMOO*s in the *PopulationMOO*.

Parameters: None.

Return Value: The mean fitness value of the *PopulationMOO* or 0.0 if the *PopulationMOO* is empty.

Caveats: None.

No. TO-PM-123

double **stdDevFitness**( ) const;

Returns the standard deviation of all fitness values of the *PopulationMOO* computed as $\sigma = \frac{1}{n}\sum_{i=0}^{n-1}\left[f(this_i) - \overline{f(this_i)}\right]^2$ with $n = |this|$. f is the function that evaluates the fitness of an *IndividualMOO*.

Parameters: None.

Return Value: Standard deviation of all fitness values or 0.0, if the *PopulationMOO* is empty.

Caveats: None.

3.11 Select one *IndividualMOO*

3.11.1 The best *IndividualMOO*

No. TO-PM-100

unsigned **bestIndex**( ) const;

Returns the index of the *IndividualMOO* with the best fitness value. The best fitness value depends on the value of the *ascending*-flag. If *ascending* is set to "true", the minimum fitness value is the best, if *ascending* is set to "false", the maximum fitness value is the winner. The *PopulationMOO* must contain at least one *IndividualMOO*, otherwise the method will be aborted with an error message. This method also works properly, when the *IndividualMOO*s are not already sorted.

Parameters: None.

Return Value: Index of the *IndividualMOO* with the best fitness value.

Caveats: None.

No. TO-PM-104

IndividualMOO& **best**( );

Same as above (method *bestIndex*), but here not only the index of the best *IndividualMOO*, but the *IndividualMOO* itself is returned.

Parameters: None.

Return Value: *IndividualMOO* with the best fitness value.

Caveats: None.

No. TO-PM-105

const IndividualMOO& **best**( ) const;

Same as above (method *bestIndex*), but here not only the index of the best *IndividualMOO*, but the *IndividualMOO* itself is returned.

Parameters: None.

Return Value: *IndividualMOO* with the best fitness value.

Caveats: None.

No. TO-PM-102

IndividualMOO& **oneOfBest**( );

Return one of *IndividualMOO*s with the best fitness value. A random is used to select one of them.

Parameters: None.

Return Value: Randomly selected *IndividualMOO* from the *IndividualMOO*s with the best fitness value.

Caveats: None.

No. TO-PM-103

const IndividualMOO& **oneOfBest**( ) const;

Return one of *IndividualMOO*s with the best fitness value. A random is used to select one of them.

Parameters: None.

Return Value: Randomly selected *IndividualMOO* from the *IndividualMOO*s with the best fitness value.

Caveats: None.

3.11.2 The worst *IndividualMOO*

No. TO-PM-101

unsigned **worstIndex**( ) const;

Returns the index of the *IndividualMOO* with the worst fitness value. The worst fitness value depends on the value of the *ascending*-flag. If *ascending* is set to "true", the maximum fitness value is the worst, if *ascending* is set to "false", the minimum fitness value is the looser. The *PopulationMOO* must contain at least one *IndividualMOO*, otherwise the method will be aborted with an error message. This method also works properly, when the *IndividualMOO*s are not already sorted.

Parameters: None.

Return Value: Index of the *IndividualMOO* with the worst fitness value.

Caveats: None.

No. TO-PM-106

IndividualMOO& **worst**( );

Same as above (method *worstIndex*), but here not only the index of the worst *IndividualMOO*, but the *IndividualMOO* itself is returned.

Parameters: None.

Return Value: *IndividualMOO* with the worst fitness value.

Caveats: None.

No. TO-PM-107

const IndividualMOO& **worst**( ) const;

Same as above (method *worstIndex*), but here not only the index of the worst *IndividualMOO*, but the *IndividualMOO* itself is returned.

Parameters: None.

Return Value: *IndividualMOO* with the worst fitness value.

Caveats: None.

3.11.3 The randomly chosen *IndividualMOO*

No. TO-PM-108

IndividualMOO& **random**( );

Returns a randomly chosen *IndividualMOO* of *this*.

Parameters: None.

Return Value: An *IndividualMOO* of *this*.

Caveats: None.

No. TO-PM-109

const IndividualMOO& **random**( ) const;

Returns a randomly chosen *IndividualMOO* of *this*.

Parameters: None.

Return Value: An *IndividualMOO* of *this*.

Caveats: None.

3.12 Selection for SOO

3.12.1 Preparation for Selection

Initialization

No. TO-PM-400

void **selectInit**( );

Initializes the internal variables of all *IndividualMOO*s of the *PopulationMOO* for a following selection method.

Parameters: None.

Return Value: None.

Caveats: None.

Select Elitists

No. TO-PM-401

void **selectElitists**(PopulationMOO& *offspring*,
unsigned *numElitists*);

Looks for *numElitists Individual*s with the best fitness values and copies them to the beginning of the *PopulationMOO this*. The selection probability of the elitists and the reminders of *IndividualMOO*s will be adapted.

Parameters:
- *offspring* - The mutated *PopulationMOO*.
- *numElitists* - Number of elitists. If *numElitists* is greater than the number of *IndividualMOO*s in *this*, then only as many elitists are sought as *this* contains *IndividualMOO*s.

Return Value: None.

Caveats: None.

Roulette Wheel

No. TO-PM-402

void **selectRouletteWheel**(	PopulationMOO&	*popmoo*,	
	unsigned	*numElitists*	);

Selects the reminders of *Individuals* after selecting elitists with a roulette wheel. The basic idea behind this selection method is a spinning roulette wheel. Depending on the value of the *spinOnce*-flag, the wheel can be spinned several times (flag = "false") which is similar to a stochastic selection or only one time (flag = "true") which is similar to a non-stochastic selection. For more information cf. [5] and [6].

Parameters: *popmoo* - *PopulationMOO* from which the reminders are chosen after selecting elitists.

numElitists - Number of elitists to be selected.

Return Value: None.

Caveats: None.

3.12.2 (μ, λ), $(\mu + \lambda)$ Selection

No. TO-PM-410

void **selectMuLambda**(	PopulationMOO&	*offspring*,	
	unsigned	*numElitists*	);

Selects *IndividualMOOs* from the mutated *PopulationMOO offspring* and/or from the current *PopulationMOO this* for reproduction. *numElitists* elite *IndividualMOOs* will be taken over from *offspring* and *this*. If *numElitists* is set to zero, then all current *IndividualMOOs* will be dismissed; this is equal to a (μ, λ)-selection. If *numElitists* greater than zero, then this number of elitists will be taken over from both *PopulationMOOs*; this is equal to a $(\mu + \lambda)$-selection. For more information cf. [7].

Parameters: *offspring* - The mutated *PopulationMOO*.

numElitists - Number of elitists to be taken over from both *PopulationMOOs*, e.g. a parent and an offspring. The default value is 0.

Return Value: None.

Caveats: None.

3.12.3 Proportional Selection

No. TO-PM-405

void **linearDynamicScaling**(vector< double >& *window*,
unsigned long *t*);

Scales the fitness values of all *IndividualMOO*s of *this* using the scaling window *window* at time *t*. For more information about scaling methods cf. [8].

Parameters: *window* - Window that is used for the scaling.
t - Time (that means position inside the window) at which is scaled.

Return Value: None.

Caveats: None.

No. TO-PM-412

void **selectProportional**(PopulationMOO& *popmoo*,
unsigned *numElitists*);

Selects *Individual*s from the mutated *PopulationMOO popmoo* and the current *PopulationMOO this* for reproduction using the proportional selection, i.e. all fitness values will be adapted in a way that they are proportional to the scaled fitness values. After that, the *IndividualMOO*s are chosen for reproduction by calling the method *selectRouletteWheel*.
For more information cf. [2].

Parameters: *popmoo* - The mutated *PopulationMOO* to be selected.
numElitists - Number of elitists to be taken over. If *numElitists* is zero, then a proportional selection takes place. If the value is greater than zero, *numElitists* elitists will be taken over from both *PopulationMOO*s and the rest will be selected from *popmoo*.

Return Value: None.

Caveats: The fitness values of all *IndividualMOO*s of the population must be positive.

No. TO-PM-413

IndividualMOO& **selectOneIndividual**( );

Selects an *IndividualMOO* for reproduction by using proportional selection, i.e. the selection probability of all *IndividualMOO*s will be adapted to scaled fitness values. After that, an *IndividualMOO* will be chosen by calling the method *selectRouletteWheel*.
For more information cf. [2].

Parameters: None.

Return Value: The selected *IndividualMOO*.

Caveats: The fitness values of all *IndividualMOO*s must be positive.

3.12.4 Ranking Selection

Linear Ranking

No. TO-PM-418

void **selectLinearRanking**(	PopulationMOO&	*offspring*,	
	double	*etaMax*,	
	unsigned	*numElitists*	);

Selects *IndividualMOO*s from the *PopulationMOO offspring* for reproduction by using the method of *Linear Ranking*, i.e. each *IndividualMOO* receives a selection probability based upon its "ranking" inside the *PopulationMOO*. The ranking is determined by the order of the *IndividualMOO*s refering to descending fitness values. After evaluating the probabilities, the *IndividualMOO*s are selected using the method *selectRouletteWheel*. Additionally *numElitists* elitists can be taken over from the current *PopulationMOO* and the mutated *PopulationMOO*.
For more information cf. [10] and [12].

Parameters:
- *offspring* - The mutated *PopulationMOO*.
- *etaMax* - Maximum reproduction rate, the default value is 1.1.
- *numElitists* - Number of elitists to be taken over from both *PopulationMOO*s. The default is set to 0.

Return Value: None.

Caveats: None.

Whitlye's Linear Ranking

No. TO-PM-419

void **selectLinearRankingWhitley**(	PopulationMOO&	*offspring*,	
	double	*a*,	
	unsigned	*numElitists*	);

Same as above, but here the evaluation of the selection probability differs.
For more information cf. [14].

Parameters:
- *offspring* - The mutated *PopulationMOO*.
- *a* - Maximum reproduction rate, the default value is set to 1.1.
- *numElitists* - Number of elititst to be taken over from both *PopulationMOO*s. The default is 0.

Return Value: None.

Caveats: None.

Uniform Ranking

No. TO-PM-416

---

void **selectUniformRanking**(PopulationMOO& *offspring*,
unsigned *numElitists*);

Selects *IndividualMOO*s from the *PopulationMOO offspring* for reproduction using the method of *selectUniformRanking*, i.e. each *IndividualMOO* receives the same selection probability and the method *selectRouletteWheel* is used for selection. Additionally *numElitists* elitits can be taken over from the current *PopulationMOO this* and the mutated *PopulationMOO offspring*. For more information cf. [2].

Parameters: *offspring* - The mutated *PopulationMOO* from which elititsts can be taken over.
numElitists - Number of elititsts. The default value is 0.

Return Value: None.

Caveats: None.

No. TO-PM-417

---

void **reproduce**(PopulationMOO& *offspring*,
unsigned *numElitists*);

Same as method *selectUniformRanking*.

Parameters: *offspring* - See above.
numElitists - See above.

Return Value: None.

Caveats: None.

3.12.5 Tournament Selection

No. TO-PM-414

void **selectTournament**(	PopulationMOO&	*offspring*,	
	unsigned	*q*,	
	unsigned	*numElitists*	);

Selects the best *IndividualMOO* out of a randomly chosen group of *q IndividualMOO*s from the mutated *PopulationMOO offspring* and takes it over to the new *PopulationMOO this*. This selection is repeated as long as there are free slots in the new *PopulationMOO*. Additionally *numElitists* elitists can be taken over from *offspring* and *this*. No correction of the selection probabilities for elitists will take place.
For more information cf. [8].

Parameters:
- *offspring* - The mutated *PopulationMOO*.
- *q* - Number of tournament opponents, the default number is 2.
- *numElitists* - Number of elitists to be taken over from both *PopulationMOO*s, the default value is 0.

Return Value: None.

Caveats: None.

3.12.6 EP-Style Tournament Selection

No. TO-PM-415

void **selectEPTournament**(	PopulationMOO&	*offspring*,	
	unsigned	*q*	);

Selects *IndividualMOO*s from the mutated *PopulationMOO offspring* and the *PopulationMOO this* by using the *EP-style Tournament Selection* by D. B. Fogel. Each *IndividualMOO* of both *PopulationMOO*s has to compete against *q* randomly chosen individuals of the *PopulationMOO*s *this* and *offspring*. An *IndividualMOO* wins a round, when it has collected more wins during the last round, i.e. when its fitness value was better than the value of its opponent. If two *IndividualMOO*s have the same number of wins, then the *IndividualMOO* with the better fitness value wins.
For more information cf. [15].

Parameters:
- *offspring* - The mutated *PopulationMOO*.
- *q* - Number of opponents for each *IndividualMOO*.

Return Value: None.

Caveats: None.

3.12.7 Other Selections

No. TO-PM-411

void **selectMuLambdaKappa**(	PopulationMOO&	*offspring*,	
	unsigned	*lifespan*,	
	unsigned	*adolescence*	);

Selects some *IndividualMOO*s from the *PopulationMOO this* and its mutated *PopulationMOO offspring* for reproduction. The "youngest" *IndividualMOO*s will be chosen first, i.e. those *IndividualMOO*s, whose age is less than the value of *adolescence*. Second, those *IndividualMOO*s will be selected, which do not belong to the set of "youngest", but whose lifespan has not reached the value *lifespan* yet. Older *IndividualMOO*s will be dismissed.

Parameters:

- *offspring* - The mutated *PopulationMOO*.
- *lifespan* - Maximum lifespan of an *IndividualMOO*, the default value is 1.
- *adolescence* - An *IndividualMOO* must have fallen short of this age to be in the set of "youngest" *IndividualMOO*s. The default value is 0.

Return Value: None.

Caveats: None.

3.13 Ranking

3.13.1 Domination

No. TO-PM-210

int **Dominate**(unsigned *i1*,
unsigned *i2*);

Checks the relationship between *IndividualMOO*s *i1* and *i2* according to the value *ascending*.

Parameters: *i1* - The index of *IndividualMOO* which is compared.
i2 - The index of *IndividualMOO* which is compared.

Return Value: the following value which shows the relationship between *IndividualMOO*s *i1* and *i2*.
3 : *i1* dominates *i2* strongly.
2 : *i1* dominates *i2* weakly.
1 : *i1* equals *i2*.
0 : The error occured in the calculation.
-1 : There is a trade-off relationship between *i1* and *i2*.
-2 : *i2* dominates *i1* weakly.
-3 : *i2* dominates *i1* strongly.

Caveats: None.

3.13.2 MOGA

Goldberg

No. TO-PM-212

void **MOGAGoldbergRank**( );

Calculates the rank of *IndividualMOO*s according to the value *ascending* by Goldberg's method and sets these values to *MOORank*.

Parameters: None.

Return Value: None.

Caveats: None.

Fonseca & Flemming

No. TO-PM-211

---

void **MOGAFonsecaRank**( );

Calculates the rank of *IndividualMOO*s according to the value *ascending* by Fonseca & Flemming's method and sets these values to *MOORank*.

Parameters: None.

Return Value: None.

Caveats: None.

3.13.3 NSGA II

No. TO-PM-213

---

void **NSGAIIRank**( );

Calculates the rank of *IndividualMOO*s according to the value *ascending* by NSGA II method and sets these values to *MOORank*. The complexity of this calculation is $o(N^2)$.

Parameters: None.

Return Value: None.

Caveats: None.

3.14 Transfer MOO to SOO

3.14.1 Rank

No. TO-PM-250

void **MOORankToFitness**( );

Transfers the data of rank *MOORank* to the variable *fitness*.
In the minimization problem, the original values of *MOORank* are copied to *fitness*.
In the maximization problem, the values R in the following equation are copied to *fitness*.
$R_i = max(MOORank_j, j \in PopulationMOO) - MOORank_i + 1$.

Parameters: None.

Return Value: None.

Caveats: None.

3.14.2 Aggregation

No. TO-PM-251

void **Aggregation**(const vector< double >& *weight*);

Calculates the weighted sum of *MOOFitness*s and sets it to *fitness*.

Parameters: *weight* - The vector which stores the weight.

Return Value: None.

Caveats: None.

No. TO-PM-252

void **SimpleSum**( );

Calculates the sum of *MOOFitness*s and sets it to *fitness*.

Parameters: None.

Return Value: None.

Caveats: None.

3.14.3 One of *MOOFitness*s

No. TO-PM-253

---

void **SimpleTransferFitness**(unsigned i);

Picks up one of *MOOFitness*s and sets it to *fitness*.

Parameters: i - The index of objective function which you want to use as single objective function.

Return Value: None.

Caveats: None.

3.15 Selection Probability

3.15.1 Preparation

No. TO-PM-450

void **NormalizeSelectProb**( );

Normailzes selection probability of all *IndividualMOO*s in *this*.

Parameters: None.

Return Value: None.

Caveats: None.

3.15.2 Michalewicz

No. TO-PM-451

void **SelectProbMichalewicz**(double c);

Calculates the selection probability of all *IndividualMOO*s in *this*.

Parameters: c - The constant c in the following equation.
$p = c \times (1.0 - c)^{r-1}$
Here r shows the rank. The default value is 0.075.

Return Value: None.

Caveats: None.

3.16 Distance on Phenotypic Fitness Space

In MOO-EALib, I prepared the following distances.

$$N_1 = \sum_i^n |x_i - y_i| \tag{3.1}$$

$$N_2 = \sqrt{\sum_i^n (x_i - y_i)^2} \tag{3.2}$$

$$N_\infty = max(|x_i - y_i|, i = 1, ..., n) \tag{3.3}$$

No. TO-PM-600

double **PhenoFitDisN1**(unsigned *i1*,
unsigned *i2*);

Calculates the N_1 distance between *IndividualMOO*s *i1* and *i2*.

Parameters: *i1* - The index of *IndividualMOO*.
i2 - The index of *IndividualMOO*.

Return Value: The N_1 distance between *IndividualMOO*s *i1* and *i2*.

Caveats: None.

No. TO-PM-601

double **PhenoFitDisN2**(unsigned *i1*,
unsigned *i2*);

Calculates the N_2 distance between *IndividualMOO*s *i1* and *i2*.

Parameters: *i1* - The index of *IndividualMOO*.
i2 - The index of *IndividualMOO*.

Return Value: The N_2 distance between *IndividualMOO*s *i1* and *i2*.

Caveats: None.

No. TO-PM-602

double **PhenoFitDisNM**(unsigned *i1*,
unsigned *i2*);

Calculates the N_∞ distance between *IndividualMOO*s *i1* and *i2*.

Parameters: *i1* - The index of *IndividualMOO*.
i2 - The index of *IndividualMOO*.

Return Value: The N_∞ distance between *IndividualMOO*s *i1* and *i2*.

Caveats: None.

3.17 Sharing

3.17.1 Niche Count

No. TO-PM-700

unsigned **NicheCountPFN1**(unsigned *i1*,
double *sr*);

Calculates Niche count with sharing radius *sr*. The return value means the number of *IndividualMOO*s within sharing radius *sr*. The distance is calculated by N_1.

Parameters: *i1* - The index of *IndividualMOO*.
sr - Sharing radius.

Return Value: Niche count with sharing radius *sr*. The distance is calculated by N_1.

Caveats: None.

No. TO-PM-701

unsigned **NicheCountPFN2**(unsigned *i1*,
double *sr*);

Calculates Niche count with sharing radius *sr*. The return value means the number of *IndividualMOO*s within sharing radius *sr*. The distance is calculated by N_2.

Parameters: *i1* - The index of *IndividualMOO*.
sr - Sharing radius.

Return Value: Niche count with sharing radius *sr*. The distance is calculated by N_2.

Caveats: None.

No. TO-PM-702

unsigned **NicheCountPFNM**(unsigned *i1*,
double *sr*);

Calculates Niche count with sharing radius *sr*. The return value means the number of *IndividualMOO*s within sharing radius *sr*. The distance is calculated by N_∞.

Parameters: *i1* - The index of *IndividualMOO*.
sr - Sharing radius.

Return Value: Niche count with sharing radius *sr*. The distance is calculated by N_∞.

Caveats: None.

No. TO-PM-703

void **NicheCountPFN1**(double *sr*);

Calculates Niche counts with sharing radius *sr* for all *IndividualMOO*s and sets them to *MOOShare*s. The distance is calculated by N_1.

Parameters: *sr* - Sharing radius.

Return Value: None.

Caveats: None.

No. TO-PM-704

void **NicheCountPFN2**(double *sr*);

Calculates Niche counts with sharing radius *sr* for all *IndividualMOO*s and sets them to *MOOShare*s. The distance is calculated by N_2.

Parameters: *sr* - Sharing radius.

Return Value: None.

Caveats: None.

No. TO-PM-705

void **NicheCountPFNM**(double *sr*);

Calculates Niche counts with sharing radius *sr* for all *IndividualMOO*s and sets them to *MOOShare*s. The distance is calculated by N_∞.

Parameters: *sr* - Sharing radius.

Return Value: None.

Caveats: None.

3.17.2 Sharing Function ($s = 1.0 - \frac{d}{\sigma_{share}}$)

No. TO-PM-710

---

double **SharingTriPFN1**(unsigned *i1*,
double *sr*);

Calculates sharing value by $1.0 - \frac{N_1}{sr}$. Here N_1 and *sr* are a distance and a sharing radius, respectively.

Parameters: *i1* - The index of *IndividualMOO*.
sr - Sharing radius.

Return Value: Sharing value by $1.0 - \frac{N_1}{sr}$.

Caveats: None.

No. TO-PM-711

---

double **SharingTriPFN2**(unsigned *i1*,
double *sr*);

Calculates sharing value by $1.0 - \frac{N_2}{sr}$. Here N_2 and *sr* are a distance and a sharing radius, respectively.

Parameters: *i1* - The index of *IndividualMOO*.
sr - Sharing radius.

Return Value: Sharing value by $1.0 - \frac{N_2}{sr}$.

Caveats: None.

No. TO-PM-712

---

double **SharingTriPFNM**(unsigned *i1*,
double *sr*);

Calculates sharing value by $1.0 - \frac{N_\infty}{sr}$. Here N_∞ and *sr* are a distance and a sharing radius, respectively.

Parameters: *i1* - The index of *IndividualMOO*.
sr - Sharing radius.

Return Value: Sharing value by $1.0 - \frac{N_\infty}{sr}$.

Caveats: None.

No. TO-PM-713

void **SharingTriPFN1**(double *sr*);

Calculates sharing values by $1.0 - \frac{N_1}{sr}$ for all *IndividualMOO*s and sets them to *MOOShare*s. Here N_1 and *sr* are a distance and a sharing radius, respectively.

Parameters: *sr* - Sharing radius.

Return Value: None.

Caveats: None.

No. TO-PM-714

void **SharingTriPFN2**(double *sr*);

Calculates sharing values by $1.0 - \frac{N_2}{sr}$ for all *IndividualMOO*s and sets them to *MOOShare*s. Here N_2 and *sr* are a distance and a sharing radius, respectively.

Parameters: *sr* - Sharing radius.

Return Value: None.

Caveats: None.

No. TO-PM-715

void **SharingTriPFNM**(double *sr*);

Calculates sharing values by $1.0 - \frac{N_\infty}{sr}$ for all *IndividualMOO*s and sets them to *MOOShare*s. Here N_∞ and *sr* are a distance and a sharing radius, respectively.

Parameters: *sr* - Sharing radius.

Return Value: None.

Caveats: None.

3.17.3 Sharing Function ($s = 1.0 - (\frac{d}{\sigma_{share}})^p$)

No. TO-PM-720

double **SharingPowPFN1**(unsigned *i1*,
double *sr*,
double *pw*);

Calculates sharing value by $1.0 - (\frac{N_1}{sr})^{pw}$. Here N_1, sr and pw are a distance, a sharing radius and an exponent, respectively.

Parameters: *i1* - The index of *IndividualMOO*.
sr - Sharing radius.
pw - The exponent.

Return Value: Sharing value by $1.0 - (\frac{N_1}{sr})^{pw}$.

Caveats: None.

No. TO-PM-721

double **SharingPowPFN2**(unsigned *i1*,
double *sr*,
double *pw*);

Calculates sharing value by $1.0 - (\frac{N_2}{sr})^{pw}$. Here N_2, sr and pw are a distance, a sharing radius and an exponent, respectively.

Parameters: *i1* - The index of *IndividualMOO*.
sr - Sharing radius.
pw - The exponent.

Return Value: Sharing value by $1.0 - (\frac{N_2}{sr})^{pw}$.

Caveats: None.

No. TO-PM-722

double **SharingPowPFNM**(unsigned *i1*,
double *sr*,
double *pw*);

Calculates sharing value by $1.0 - (\frac{N_\infty}{sr})^{pw}$. Here N_∞, sr and pw are a distance, a sharing radius and an exponent, respectively.

Parameters: *i1* - The index of *IndividualMOO*.
sr - Sharing radius.
pw - The exponent.

Return Value: Sharing value by $1.0 - (\frac{N_\infty}{sr})^{pw}$.

Caveats: None.

No. TO-PM-723

void **SharingPowPFN1**(double *sr*,
double *pw*);

Calculates sharing values by $1.0 - (\frac{N_1}{sr})^{pw}$ for all *IndividualMOO*s and sets them to *MOOShare*s. Here N_1, *sr* and *pw* are a distance, a sharing radius and an exponent, respectively.

Parameters: *sr* - Sharing radius.
pw - The exponent.

Return Value: None.

Caveats: None.

No. TO-PM-724

void **SharingPowPFN2**(double *sr*,
double *pw*);

Calculates sharing values by $1.0 - (\frac{N_2}{sr})^{pw}$ for all *IndividualMOO*s and sets them to *MOOShare*s. Here N_2, *sr* and *pw* are a distance, a sharing radius and an exponent, respectively.

Parameters: *sr* - Sharing radius.
pw - The exponent.

Return Value: None.

Caveats: None.

No. TO-PM-725

void **SharingPowPFNM**(double *sr*,
double *pw*);

Calculates sharing values by $1.0 - (\frac{N_\infty}{sr})^{pw}$ for all *IndividualMOO*s and sets them to *MOOShare*s. Here N_∞, *sr* and *pw* are a distance, a sharing radius and an exponent, respectively.

Parameters: *sr* - Sharing radius.
pw - The exponent.

Return Value: None.

Caveats: None.

3.17.4 Share values

No. TO-PM-790

void **SharingSelProb**( );

Shares a selection probability. The calculation will be done by $p_{share} = \frac{p}{s}$. Here p_{share}, p and s are a shared selection probability, an original selection probability and a sharing value (or Niche count). After this calculation, normalization of selection probability should be done.

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-PM-791

void **SharingFitness**( );

Shares a fitness value for SOO. The calculation will be done by $f_{share} = \frac{f}{s}$. Here f_{share}, f and s are a shared fitness value, an original fitness value and a sharing value (or Niche count).

Parameters: None.

Return Value: None.

Caveats: None.

No. TO-PM-792

void **SharingMOOFitness**( );

Shares fitness values for MOO. The calculation will be done by $f_{share} = \frac{f}{s}$. Here f_{share}, f and s are a shared fitness value, an original fitness value and a sharing value (or Niche count).

Parameters: None.

Return Value: None.

Caveats: None.

3.18 Selection for MOO

3.18.1 Preparation

No. TO-PM-800

void **SelectByRoulette**(	PopulationMOO&	*offsprings*,	
	unsigned	*elit*	);

Selects *IndividualMOO*s after selecting *elit* elitists with a roulette wheel.

Parameters: *offsprings* - The mutated *PopulationMOO*.

elit - The number of elitists which have been selected already.

Return Value: None.

Caveats: None.

3.18.2 Tournament Selection

No. TO-PM-801

void **SelectTournamentNCPFN1**(	PopulationMOO&	*offsprings*,	
	double	*share*,	
	unsigned	*elit*	);

Selects *IndividualMOO*s after selecting *elit* elitsits with Tournament selection. Two *IndividualMOO*s are selected randomly. If one *IndividualMOO* dominates the other, then this *IndividualMOO* is selected. If the trade-off relationship among *IndividualMOO*s exists, the *IndividualMOO* with small niche count is selected. If niche counts are the same, *IndividualMOO* is selected randomly. N_1 distance is used on Phenotypic Fitness space to calculate the distance.

Parameters: *offsprings* - The mutated *PopulationMOO*.

share - The sharing radius.

elit - The number of elitists which have been selected already.

Return Value: None.

Caveats: NCPFN1 means "Niche Count", "Phenotypic Fitness space" and "N_1 distance".

No. TO-PM-802

void **SelectTournamentNCPFN2**(PopulationMOO& *offsprings*,
double *share*,
unsigned *elit*);

See the function whose number is "TO-PM-801". N_2 distance is used on Phenotypic Fitness space to calculate the distance.

Parameters: *offsprings* - The mutated *PopulationMOO*.
share - The sharing radius.
elit - The number of elitists which have been selected already.

Return Value: None.

Caveats: NCPFN2 means "Niche Count", "Phenotypic Fitness space" and "N_2 distance".

No. TO-PM-803

void **SelectTournamentNCPFNM**(PopulationMOO& *offsprings*,
double *share*,
unsigned *elit*);

See the function whose number is "TO-PM-801". N_∞ distance is used on Phenotypic Fitness space to calculate the distance.

Parameters: *offsprings* - The mutated *PopulationMOO*.
share - The sharing radius.
elit - The number of elitists which have been selected already.

Return Value: None.

Caveats: NCPFNM means "Niche Count", "Phenotypic Fitness space" and "N_∞ distance".

No. TO-PM-804

void **SelectTournamentSTPFN1**(	PopulationMOO&	*offsprings*,	
	double	*share*,	
	unsigned	*elit*	);

Selects *IndividualMOO*s after selecting *elit* elitsits with Tournament selection. Two *IndividualMOO*s are selected randomly. If one *IndividualMOO* dominates the other, then this *IndividualMOO* is selected. If the trade-off relationship among *IndividualMOO*s exists, the *IndividualMOO* with small sharing value is selected. If sharing values are the same, *IndividualMOO* is selected randomly. The sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$ and N_1 distance are used on Phenotypic Fitness space to calculate the distance.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *share* - The sharing radius.
- *elit* - The number of elitists which have been selected already.

Return Value: None.

Caveats: STPFN1 means "Sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$", "Phenotypic Fitness space" and "N_1 distance".

No. TO-PM-805

void **SelectTournamentSTPFN2**(	PopulationMOO&	*offsprings*,	
	double	*share*,	
	unsigned	*elit*	);

See the function whose number is "TO-PM-804". The sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$ and N_2 distance are used on Phenotypic Fitness space to calculate the distance.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *share* - The sharing radius.
- *elit* - The number of elitists which have been selected already.

Return Value: None.

Caveats: STPFN2 means "Sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$", "Phenotypic Fitness space" and "N_2 distance".

No. TO-PM-806

void **SelectTournamentSTPFNM**(	PopulationMOO&	*offsprings*,	
	double	*share*,	
	unsigned	*elit*	);

See the function whose number is "TO-PM-804". The sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$ and N_∞ distance are used on Phenotypic Fitness space to calculate the distance.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *share* - The sharing radius.
- *elit* - The number of elitists which have been selected already.

Return Value: None.

Caveats: STPFNM means "Sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$", "Phenotypic Fitness space" and "N_∞ distance".

No. TO-PM-807

void **SelectTournamentSPPFN1**(	PopulationMOO&	*offsprings*,	
	double	*share*,	
	double	*pow*,	
	unsigned	*elit*	);

Selects *IndividualMOO*s after selecting *elit* elitsits with Tournament selection. Two *IndividualMOO*s are selected randomly. If one *IndividualMOO* dominates the other, then this *IndividualMOO* is selected. If the trade-off relationship among *IndividualMOO*s exists, the *IndividualMOO* with small sharing value is selected. If sharing values are the same, *IndividualMOO* is selected randomly. The sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$ and N_1 distance are used on Phenotypic Fitness space to calculate the distance.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *share* - The sharing radius.
- *pow* - The exponent in the sharing function.
- *elit* - The number of elitists which have been selected already.

Return Value: None.

Caveats: SPPFN1 means "Sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$", "Phenotypic Fitness space" and "N_1 distance".

No. TO-PM-808

void **SelectTournamentSPPFN2**(PopulationMOO& *offsprings*,
double *share*,
double *pow*,
unsigned *elit*);

See the function whose number is "TO-PM-807". The sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$ and N_2 distance are used on Phenotypic Fitness space to calculate the distance.

Parameters:
offsprings - The mutated *PopulationMOO*.
share - The sharing radius.
pow - The exponent in the sharing function.
elit - The number of elitists which have been selected already.

Return Value: None.

Caveats: SPPFN2 means "Sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$", "Phenotypic Fitness space" and "N_2 distance".

No. TO-PM-809

void **SelectTournamentSPPFNM**(PopulationMOO& *offsprings*,
double *share*,
double *pow*,
unsigned *elit*);

See the function whose number is "TO-PM-807". The sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$ and N_∞ distance are used on Phenotypic Fitness space to calculate the distance.

Parameters:
offsprings - The mutated *PopulationMOO*.
share - The sharing radius.
pow - The exponent in the sharing function.
elit - The number of elitists which have been selected already.

Return Value: None.

Caveats: SPPFNM means "Sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$", "Phenotypic Fitness space" and "N_∞ distance".

3.18.3 Tournament Selection with a comparison set

No. TO-PM-810

void **SelectComparisonNCPFN1**(	PopulationMOO&	*offsprings*,	
	unsigned	*nc*,	
	double	*share*,	
	unsigned	*elit*	);

Selects *IndividualMOO*s after selecting *elit* elitsits. Two *IndividualMOO*s are selected randomly and also *nc IndividualMOO*s are selected randomly as a comparison set. If one *IndividualMOO* dominates all *IndividualMOO*s in the comparison set and the other doesn't, then this *IndividualMOO* is selected. In the other cases, the *IndividualMOO* with small niche count is selected. If niche counts are the same, *IndividualMOO* is selected randomly. N_1 distance is used on Phenotypic Fitness space to calculate the distance.

Parameters:

offsprings	-	The mutated *PopulationMOO*.
nc	-	The number of *IndividualMOO*s in the comparison set.
share	-	The sharing radius.
elit	-	The number of elitists which have been selected already.

Return Value: None.

Caveats: NCPFN1 means "Niche Count", "Phenotypic Fitness space" and "N_1 distance".

No. TO-PM-811

void **SelectComparisonNCPFN2**(	PopulationMOO&	*offsprings*,	
	unsigned	*nc*,	
	double	*share*,	
	unsigned	*elit*	);

See the function whose number is "TO-PM-810". N_2 distance is used on Phenotypic Fitness space to calculate the distance.

Parameters:

offsprings	-	The mutated *PopulationMOO*.
nc	-	The number of *IndividualMOO*s in the comparison set.
share	-	The sharing radius.
elit	-	The number of elitists which have been selected already.

Return Value: None.

Caveats: NCPFN2 means "Niche Count", "Phenotypic Fitness space" and "N_2 distance".

No. TO-PM-812

void **SelectComparisonNCPFNM**(	PopulationMOO&	*offsprings*,	
	unsigned	*nc*,	
	double	*share*,	
	unsigned	*elit*	);

See the function whose number is "TO-PM-810". N_∞ distance is used on Phenotypic Fitness space to calculate the distance.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *nc* - The number of *IndividualMOO*s in the comparison set.
- *share* - The sharing radius.
- *elit* - The number of elitists which have been selected already.

Return Value: None.

Caveats: NCPFNM means "Niche Count", "Phenotypic Fitness space" and "N_∞ distance".

No. TO-PM-813

void **SelectComparisonSTPFN1**(	PopulationMOO&	*offsprings*,	
	unsigned	*nc*,	
	double	*share*,	
	unsigned	*elit*	);

Selects *IndividualMOO*s after selecting *elit* elitsits. Two *IndividualMOO*s are selected randomly and also *nc* *IndividualMOO*s are selected randomly as a comparison set. If one *IndividualMOO* dominates all *IndividualMOO*s in the comparison set and the other doesn't, then this *IndividualMOO* is selected. In the other cases, the *IndividualMOO* with small sharing value is selected. If sharing values are the same, *IndividualMOO* is selected randomly. The sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$ and N_1 distance is used on Phenotypic Fitness space to calculate the distance.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *nc* - The number of *IndividualMOO*s in the comparison set.
- *share* - The sharing radius.
- *elit* - The number of elitists which have been selected already.

Return Value: None.

Caveats: STPFN1 means "Sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$", "Phenotypic Fitness space" and "N_1 distance".

No. TO-PM-814

void **SelectComparisonSTPFN2**(PopulationMOO& *offsprings*,
unsigned *nc*,
double *share*,
unsigned *elit*);

See the function whose number is "TO-PM-813". The sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$ and N_2 distance is used on Phenotypic Fitness space to calculate the distance.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *nc* - The number of *IndividualMOO*s in the comparison set.
- *share* - The sharing radius.
- *elit* - The number of elitists which have been selected already.

Return Value: None.

Caveats: STPFN2 means "Sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$", "Phenotypic Fitness space" and "N_2 distance".

No. TO-PM-815

void **SelectComparisonSTPFNM**(PopulationMOO& *offsprings*,
unsigned *nc*,
double *share*,
unsigned *elit*);

See the function whose number is "TO-PM-813". The sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$ and N_∞ distance is used on Phenotypic Fitness space to calculate the distance.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *nc* - The number of *IndividualMOO*s in the comparison set.
- *share* - The sharing radius.
- *elit* - The number of elitists which have been selected already.

Return Value: None.

Caveats: STPFNM means "Sharing function $s = 1.0 - \frac{d}{\sigma_{share}}$", "Phenotypic Fitness space" and "N_∞ distance".

void **SelectComparisonSPPFN1**(	PopulationMOO&	*offsprings*,	
	unsigned	*nc*,	
	double	*share*,	
	double	*pow*,	
	unsigned	*elit*	);

Selects *IndividualMOO*s after selecting *elit* elitsits. Two *IndividualMOO*s are selected randomly and also *nc IndividualMOO*s are selected randomly as a comparison set. If one *IndividualMOO* dominates all *IndividualMOO*s in the comparison set and the other doesn't, then this *IndividualMOO* is selected. In the other cases, the *IndividualMOO* with small sharing value is selected. If sharing values are the same, *IndividualMOO* is selected randomly. The sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$ and N_1 distance is used on Phenotypic Fitness space to calculate the distance.

Parameters:

offsprings	-	The mutated *PopulationMOO*.
nc	-	The number of *IndividualMOO*s in the comparison set.
share	-	The sharing radius.
pow	-	The exponent in the sharing function.
elit	-	The number of elitists which have been selected already.

Return Value: None.

Caveats: SPPFN1 means "Sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$", "Phenotypic Fitness space" and "N_1 distance".

No. TO-PM-817

void **SelectComparisonSPPFN2**(PopulationMOO& *offsprings*,
unsigned *nc*,
double *share*,
double *pow*,
unsigned *elit*);

See the function whose number is "TO-PM-816". The sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$ and N_2 distance is used on Phenotypic Fitness space to calculate the distance.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *nc* - The number of *IndividualMOO*s in the comparison set.
- *share* - The sharing radius.
- *pow* - The exponent in the sharing function.
- *elit* - The number of elitists which have been selected already.

Return Value: None.

Caveats: SPPFN2 means "Sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$", "Phenotypic Fitness space" and "N_2 distance".

No. TO-PM-818

void **SelectComparisonSPPFNM**(PopulationMOO& *offsprings*,
unsigned *nc*,
double *share*,
double *pow*,
unsigned *elit*);

See the function whose number is "TO-PM-816". The sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$ and N_∞ distance is used on Phenotypic Fitness space to calculate the distance.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *nc* - The number of *IndividualMOO*s in the comparison set.
- *share* - The sharing radius.
- *pow* - The exponent in the sharing function.
- *elit* - The number of elitists which have been selected already.

Return Value: None.

Caveats: SPPFNM means "Sharing function $s = 1.0 - (\frac{d}{\sigma_{share}})^p$", "Phenotypic Fitness space" and "N_∞ distance".

3.18.4 Elitists for MOO

In MOO-EALib, I used the following words:

elitist : The *IndividualMOO* with Rank 1 and niche count (sharing value) 1.

sub-elitist : The *IndividualMOO* with Rank 1.

It is clear that the set of *sub-elitist*s includes the whole set of *elitist*s.

No. TO-PM-850

---

bool **EvaluationMOOElitists**(unsigned *i1*);

Evaluates *i1*th *IndividualMOO*. If this *individualMOO* is an elitist, "True" will be returned. If not, "False" will be returned.

Parameters: *i1* - The index of *IndividualMOO* which is evaluated.

Return Value: *True* : This *IndividualMOO* is an elitist.
False : This *IndividualMOO* is not an elitist.

Caveats: None.

No. TO-PM-851

---

unsigned **getNoOfRankOne**( );

Gets the number of *IndividualMOO*s with Rank 1.

Parameters: None.

Return Value: The number of *IndividualMOO*s with Rank 1.

Caveats: None.

No. TO-PM-852

unsigned **SelectMOOElitists**(PopulationMOO& *offsprings*,
unsigned *MOR*,
unsigned *MOS*,
double *SR*,
double *POW*);

Selects elitists and gets the number of *IndividualMOO*s which are selected as elitists.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *MOR* - The function number (211 - 213) for Rank.
- *MOS* - The function number (703 - 725) for sharing.
- *SR* - The sharing radius.
- *POW* - The exponent in the sharing function. If powed sharing function is used, this value is valid.

Return Value: The number of *IndividualMOO*s which are selected as elitists.

Caveats: None.

No. TO-PM-853

unsigned **SelectMOOElitistsSub**(PopulationMOO& *offsprings*,
unsigned *MOR*,
unsigned *MOS*,
double *SR*,
double *POW*);

Selects sub-elitists and gets the number of *IndividualMOO*s which are selected as sub-elitists.

Parameters:
- *offsprings* - The mutated *PopulationMOO*.
- *MOR* - The function number (211 - 213) for Rank.
- *MOS* - The function number (703 - 725) for sharing.
- *SR* - The sharing radius.
- *POW* - The exponent in the sharing function. If powed sharing function is used, this value is valid.

Return Value: The number of *IndividualMOO*s which are selected as sub-elitists.

Caveats: None.

Chapter 4

Conclusion

In this report, we explained the functions in *MOO-EALib*. Chapter 2 and 3 can be used as the manual of *MOO-EALib*.

Now, we are concluding this report. But, the development of *MOO-EALib* is ongoing. Every day, some functions are added in *MOO-EALib* and some functions are updated. Of course, if some functions have bugs, then they will be corrected. Thus, this report is on our way to our final goal. Probably, some additional reports are necessary in the near future to support new functions. In that time, we will write some additional reports.

We will promise them !!!

Acknowledgement

Before finishing this report, we would like to thank following researchers.

1. Christian Igel (Ruhr University Bochum) for his technical adivice about EALib.
2. Marc-Oliver Gewaltig (HONDA R&D Europe) for his technical advice about C++.
3. Markus Olhofer (HONDA R&D Europe) for his technical advice.
4. Bernhard Sendhoff (HONDA R&D Europe) for his technical advice.
5. EL-TEC group (HONDA R&D Europe) for their useful discussions.

Bibliography

[1] Tatsuya Okabe, MOO-EALib (Part 1) : Structure, HONDA R& D Europe (Deutschland) GmbH Internal Report HRE-G/FTR Report 01/09, 2001

[2] Martin Kreutz, Bernhard Sendhoff and Christian Igel, EALib: A C++ class library for evolutionary algorithms version 1.5, 2000

[3] Rüdiger Alberts, Quick Reference for the *EA*-Library, 2000

[4] Rüdiger Alberts, Reference for the *EA*-Library, 2000

[5] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[6] K.A.D. Jong, An Analysis of the Behaviour of a Class of Genetic Adaptive Systems, Dept. Computer and Communication Sciences, University of Michigan, 1975, Diss. Abstr. Int. 36(10), 5140B, University Microfilms No. 76-9381.

[7] T. Bäck, Evolutionary Algorithms in Theory and Practice, dissertation thesis, University of Dortmund, Department of Computer Science, February 1994.

[8] D.E. Goldberg and K. Deb, A comparative analysis of selection schemes used in genetic algorithms, p69 - 93 in [9].

[9] G.J.E. Rawlings (Editor), Foundations of Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[10] J.E. Baker, Adaptive Selection methods for genetic algorithms, p101 - 111 in [11].

[11] Proceedings of the First International Conference on Genetic Algorithms and Their Applications (ICGA 1985), J.J. Grefenstette, Pittsburgh, PA, Juli 24-26 1985, Carnegie-Mellon University, Lawrence Erlbaum Associates (Hillsdale, NJ, 1988).

[12] J.J. Grefenstette and J.E. Baker, How genetic algorithms work: A critical look at implicit parallelism, p20 - 27 in [13].

[13] Proceedings of the Third International Conference on Genetic Algorithms (ICGA 1989), J.D. Schaffer, July 4-7, San Mateo, CA, 1989, George Mason University, Morgan Kaufmann Publishers Inc.

[14] L.D. Whitley, The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best, p116 - 121 in [13].

[15] D.B. Fogel, Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press, 1995.

Index