Project 3 - Evolutionary Algorithms

3.0

Generated by Doxygen 1.8.15

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 DEA_Config Struct Reference

Holds all the user defined variables. Differential Evolution Algorithm Configuration Structure, where all user defined variables that are used to configure the Differential Evolution Algorithm are stored.

```
#include <DifferentialEvolution.h>
```

Public Attributes

- int dimensions
- int NP
- int generations
- double cr
- double f
- double lambda
- int strategy

3.1.1 Detailed Description

Holds all the user defined variables. Differential Evolution Algorithm Configuration Structure, where all user defined variables that are used to configure the Differential Evolution Algorithm are stored.

3.1.2 Member Data Documentation

3.1.2.1 cr

```
double DEA_Config::cr
```

Crossover Probability.

3.1.2.2 dimensions

`int DEA_Config::dimensions`

Number of dimensions per individual in population.

3.1.2.3 f

`double DEA_Config::f`

A scaling factor.

3.1.2.4 generations

`int DEA_Config::generations`

Maximum number of generations.

3.1.2.5 lambda

`double DEA_Config::lambda`

A scaling factor.

3.1.2.6 NP

`int DEA_Config::NP`

Population size.

3.1.2.7 strategy

`int DEA_Config::strategy`

The Differential Evolution strategy to use for mutation/crossover.

The documentation for this struct was generated from the following file:

- C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/DifferentialEvolution.h

3.2 DEA_Population Struct Reference

Holds all the population information. Differential Evolution Algorithm Population Structure, holds all the data related to the population of the Differential Evolution Algorithm.

`#include <DifferentialEvolution.h>`

Public Attributes

- int functionID
- vector< double > bounds
- int functionCounter = 0
- vector< double > fitness
- vector< vector< double > > pop
- vector< double > bestGenFitness
- double executionTime = -1.0

3.2.1 Detailed Description

Holds all the population information. Differential Evolution Algorithm Population Structure, holds all the data related to the population of the Differential Evolution Algorithm.

3.2.2 Member Data Documentation

3.2.2.1 bestGenFitness

```
vector<double> DEA_Population::bestGenFitness
```

Keeps track of best fitness from each generation.

3.2.2.2 bounds

```
vector<double> DEA_Population::bounds
```

Holds the (min,max) bounds of the values for each individual in the population.

3.2.2.3 executionTime

```
double DEA_Population::executionTime = -1.0
```

Time(ms) it took to run the Genetic Algorithm on this population.

3.2.2.4 fitness

```
vector<double> DEA_Population::fitness
```

The fitness for each vector in the population matrix.

3.2.2.5 functionCounter

```
int DEA_Population::functionCounter = 0
```

The function counter keeps track of how many times the benchmark function was called.

3.2.2.6 functionID

```
int DEA_Population::functionID
```

The ID determines which benchmark function to call.

3.2.2.7 pop

```
vector<vector<double> > DEA_Population::pop
```

The population matrix.

The documentation for this struct was generated from the following file:

- C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/DifferentialEvolution.h

3.3 DEAAnalysis Struct Reference

Differential Evolution Algorithm Analysis Differential Evolution Algorithm Analysis Structure, to keep track of the analysis performed on each population in the population list.

```
#include <DifferentialEvolution.h>
```

Public Attributes

- string header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time(ms),Function Calls,Strategy\n"
- vector< int > functionIDs
- vector< double > avgFunctionFitness
- vector< double > standardDeviation
- vector< vector< double > > ranges
- vector< double > medianFunctionFitness
- vector< double > executionTimes
- vector< int > functionCalls

3.3.1 Detailed Description

Differential Evolution Algorithm Analysis Differential Evolution Algorithm Analysis Structure, to keep track of the analysis performed on each population in the population list.

3.3.2 Member Data Documentation

3.3.2.1 avgFunctionFitness

```
vector<double> DEAAnalysis::avgFunctionFitness
```

List of the average fitness per FunctionData structure.

3.3.2.2 executionTimes

```
vector<double> DEAAnalysis::executionTimes
```

List of execution times in ms for all functions.

3.3.2.3 functionCalls

```
vector<int> DEAAnalysis::functionCalls
```

List of the amount of times a function was called.

3.3.2.4 functionIDs

```
vector<int> DEAAnalysis::functionIDs
```

List of function IDs.

3.3.2.5 header

```
string DEAAnalysis::header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time
Calls,Strategy\n"
```

Header used when saving the data.

3.3.2.6 medianFunctionFitness

```
vector<double> DEAAnalysis::medianFunctionFitness
```

List of the Median fitness from each FunctionData structure.

3.3.2.7 ranges

```
vector<vector<double> > DEAAnalysis::ranges
```

List of ranges for each fitness result in resultsOfFunctions.

3.3.2.8 standardDeviation

`vector<double> DEAAnalysis::standardDeviation`

List of standard fitness deviations.

The documentation for this struct was generated from the following file:

- C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/DifferentialEvolution.h

3.4 DifferentialEvolution Class Reference

Public Member Functions

- DifferentialEvolution (int dim, int sol, int gen, double cr, double f, double lambda, int strategy)
  The only constructor available to initialize the Differential Evolution Algorithm.
- double runDifferentialEvolution (int functionID, double minBound, double maxBound)
- void analyzeDEAResults ()
  Analyzes the results of the Differential Evolution Algorithm.
- void printDEAResults ()
  Prints the Results of the Differential Evolution Algorithm.
- void printDEAAnalysis ()
  Prints the Analysis of the Differential Evolution Algorithm.
- void saveDEAResults ()
  Saves all Differential Evolution Algorithm Results to file.
- void saveDEAAnalysis ()
  Saves the Analysis of the Differential Evolution Algorithm to file.

Private Member Functions

- void generateRandDEAPopulation (DEA_Population &population)
  Generates the initial population for Differential Evolution Algorithm.
- void evaluatePopulation (int functionID, vector< vector< double >> &pop, vector< double > &fitness, int &functionCounter)
  Calculates fitness of all solutions in population.
- void evaluateIndividual (const int &functionID, vector< double > &indiv, double &fitness, int &functionCounter)
  Calculate the fitness of an individual solution of the population.
- vector< double > mutateAndCrossover (const double &cr, const double &f, const double &lambda, vector< double > &x, vector< vector< double >> indiv, mt19937 &randGenerator)
  Mutate and Crossover produces one new individual.
- void select (int functionID, vector< double > &x, double &fitness, vector< double > &newSol, int &function←Counter)
  Selects a new individual to be part of the next generation.
- void saveBestFitness (DEA_Population &pop)
  Saves the best fitness from the population.

Private Attributes

- DEA_Config **deConfig**
- vector< DEA_Population > **popList**
- DEAAnalysis **deAnalysis**

3.4.1 Constructor & Destructor Documentation

3.4.1.1 DifferentialEvolution()

```
DifferentialEvolution::DifferentialEvolution (
             int dim,
             int sol,
             int gen,
             double cr,
             double f,
             double lambda,
             int strategy )
```

The only constructor available to initialize the Differential Evolution Algorithm.

Note

No Default (zero-param) Constructor exists.

Parameters

dim	The number of dimensions each individual in the population has.
sol	The population size.
gen	The max number of generations possible.
cr	The crossover probability.
f	A scaling factor.
lambda	A scaling factor.
strategy	The Differential Evolution strategy to use.

3.4.2 Member Function Documentation

3.4.2.1 analyzeDEAResults()

```
void DifferentialEvolution::analyzeDEAResults ( )
```

Analyzes the results of the Differential Evolution Algorithm.

Analyzes the results of the Differential Evolution Algorithm.

3.4.2.2 evaluateIndividual()

```
void DifferentialEvolution::evaluateIndividual (
            const int & functionID,
            vector< double > & indiv,
            double & fitness,
            int & functionCounter )  [private]
```

Calculate the fitness of an individual solution of the population.

Calculate the fitness of an individual solution of the population.

Note

Makes function call to EA_Utilities.h --> calculateFitnessOfVector().

Parameters

functionID	The ID of the benchmark function to use.
indiv	The individual of the population.
fitness	The fitness variable for the individual.
functionCounter	A counter to keep track of how many times fitness function was called.

3.4.2.3 evaluatePopulation()

```
void DifferentialEvolution::evaluatePopulation (
            int functionID,
            vector< vector< double >> & pop,
            vector< double > & fitness,
            int & functionCounter )  [private]
```

Calculates fitness of all solutions in population.

Calculates fitness of all solutions in population.

Note

Makes function call to EA_Utilities.h --> calculateFitnessOfVector().

Parameters

functionID	The ID of the benchmark function to use.
pop	The matrix population of the Differential Evolution Algorithm.
fitness	The fitness vector for each solution from the population.
functionCounter	A counter to keep track of how many times fitness function was called.

3.4.2.4 generateRandDEAPopulation()

```
void DifferentialEvolution::generateRandDEAPopulation (
            DEA_Population & population )  [private]
```

Generates the initial population for Differential Evolution Algorithm.

Generates the initial population for Differential Evolution Algorithm.

Parameters

population	The population.

3.4.2.5 mutateAndCrossover()

```
vector< double > DifferentialEvolution::mutateAndCrossover (
            const double & cr,
            const double & f,
            const double & lambda,
            vector< double > & x,
            vector< vector< double >> indiv,
            mt19937 & randGenerator )  [private]
```

Mutate and Crossover produces one new individual.

Mutate and Crossover produces one new individual.

Note

Makes function call to DEA_Strategies.h.
indiv list of solution has the best solution at index 0 and 5 random solutions at indices 1 - 5.

Parameters

cr	The crossover probability.
f	A scaling factor.
lambda	A scaling factor.
x	The initial solution of the population.
indiv	A list of the best solution and 5 random solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new potential individual of the population.

3.4.2.6 printDEAAnalysis()

```
void DifferentialEvolution::printDEAAnalysis ( )
```

Prints the Analysis of the Differential Evolution Algorithm.

Prints the Analysis of the Differential Evolution Algorithm.

3.4.2.7 printDEAResults()

```
void DifferentialEvolution::printDEAResults ( )
```

Prints the Results of the Differential Evolution Algorithm.

Prints the Results of the Differential Evolution Algorithm.

3.4.2.8 runDifferentialEvolution()

```
double DifferentialEvolution::runDifferentialEvolution (
            int functionID,
            double minBound,
            double maxBound )
```

Runs the Differential Evolution Algorithm with set parameters.

Runs the Differential Evolution Algorithm with a set of parameters.

Parameters

functionID	The ID of the benchmark function to use.
minBound,maxBound	The minimum and maximum bounds of the population.

Returns

Returns the best result of the Differential Evolution Algorithm.

3.4.2.9 saveBestFitness()

```
void DifferentialEvolution::saveBestFitness (
            DEA_Population & pop )  [private]
```

Saves the best fitness from the population.

Saves the best fitness from the population.

Note

The best fitness is at the top (index 0) of the population.

Parameters

pop	

3.4.2.10 saveDEAAnalysis()

```
void DifferentialEvolution::saveDEAAnalysis ( )
```

Saves the Analysis of the Differential Evolution Algorithm to file.

Saves the Analysis of the Differential Evolution Algorithm to file.

3.4.2.11 saveDEAResults()

```
void DifferentialEvolution::saveDEAResults ( )
```

Saves all Differential Evolution Algorithm Results to file.

Saves all Differential Evolution Algorithm Results to file.

3.4.2.12 select()

```
void DifferentialEvolution::select (
            int functionID,
            vector< double > & x,
            double & fitness,
            vector< double > & newSol,
            int & functionCounter )  [private]
```

Selects a new individual to be part of the next generation.

Selects a new individual to be part of the next generation.

Parameters

functionID	The ID of the benchmark function to use.
x	The original individual (solution) of the population.
fitness	The fitness of the original individual of the population.
newSol	A new potential individual of the population.
functionCounter	A counter to keep track of how many times fitness function was called.

The documentation for this class was generated from the following files:

- C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/DifferentialEvolution.h
- C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/DifferentialEvolution.cpp

3.5 GA_Config Struct Reference

Holds all the user defined variables. Genetic Algorithm Configuration Structure, where all user defined variables that are used to configure the Genetic Algorithm are stored.

```
#include <GeneticAlgorithm.h>
```

Public Attributes

- int dimensions
- int solutions
- int generations
- double cr
- double mutProb
- double mutRange
- double mutPrec
- double er
- int eliteIndex
- int selectionID
- int crPoints

3.5.1 Detailed Description

Holds all the user defined variables. Genetic Algorithm Configuration Structure, where all user defined variables that are used to configure the Genetic Algorithm are stored.

3.5.2 Member Data Documentation

3.5.2.1 cr

```
double GA_Config::cr
```

Crossover Probability.

3.5.2.2 crPoints

```
int GA_Config::crPoints
```

The number of crossover points.

3.5.2.3 dimensions

```
int GA_Config::dimensions
```

Number of dimensions per individual in population.

3.5.2.4 eliteIndex

```
int GA_Config::eliteIndex
```

Elitism ending Index in the population (0 - eliteIndex).

3.5.2.5 er

```
double GA_Config::er
```

Elitism Rate.

3.5.2.6 generations

```
int GA_Config::generations
```

Maximum number of generations.

3.5.2.7 mutPrec

```
double GA_Config::mutPrec
```

Mutation Precision.

3.5.2.8 mutProb

```
double GA_Config::mutProb
```

Mutation Probability.

3.5.2.9 mutRange

```
double GA_Config::mutRange
```

Mutation Range.

3.5.2.10 selectionID

```
int GA_Config::selectionID
```

The selection type to use for the Genetic Algorithm.

3.5.2.11 solutions

```
int GA_Config::solutions
```

Population size.

The documentation for this struct was generated from the following file:

- C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/GeneticAlgorithm.h

3.6 GA_Population Struct Reference

Holds all the population information. Genetic Algorithm Population Structure, holds all the data related to the population of the Genetic Algorithm.

```
#include <GeneticAlgorithm.h>
```

Public Attributes

- int functionID
- vector< double > bounds
- int functionCounter = 0
- vector< double > fitness
- vector< vector< double > > pop
- vector< double > bestGenFitness
- double totalFitness
- double executionTime = -1.0

3.6.1 Detailed Description

Holds all the population information. Genetic Algorithm Population Structure, holds all the data related to the population of the Genetic Algorithm.

3.6.2 Member Data Documentation

3.6.2.1 bestGenFitness

```
vector<double> GA_Population::bestGenFitness
```

Keeps track of best fitness from each generation.

3.6.2.2 bounds

```
vector<double> GA_Population::bounds
```

Holds the (min,max) bounds of the values for each individual in the population.

3.6.2.3 executionTime

```
double GA_Population::executionTime = -1.0
```

Time(ms) it took to run the Genetic Algorithm on this population.

3.6.2.4 fitness

```
vector<double> GA_Population::fitness
```

The fitness for each vector in the population matrix.

3.6.2.5 functionCounter

```
int GA_Population::functionCounter = 0
```

The function counter keeps track of how many times the benchmark function was called.

3.6.2.6 functionID

```
int GA_Population::functionID
```

The ID determines which benchmark function to call.

3.6.2.7 pop

```
vector<vector<double> > GA_Population::pop
```

The population matrix.

3.6.2.8 totalFitness

```
double GA_Population::totalFitness
```

The summed up total of the fitness values.

The documentation for this struct was generated from the following file:

- C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/GeneticAlgorithm.h

3.7 GAAnalysis Struct Reference

Genetic Algorithm Analysis Genetic Algorithm Analysis Structure, to keep track of the analysis performed on each population in the population list.

```
#include <GeneticAlgorithm.h>
```

Public Attributes

- string header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time(ms),Function Calls\n"
- vector< int > functionIDs
- vector< double > avgFunctionFitness
- vector< double > standardDeviation
- vector< vector< double > > ranges
- vector< double > medianFunctionFitness
- vector< double > executionTimes
- vector< int > functionCalls

3.7.1 Detailed Description

Genetic Algorithm Analysis Genetic Algorithm Analysis Structure, to keep track of the analysis performed on each population in the population list.

3.7.2 Member Data Documentation

3.7.2.1 avgFunctionFitness

```
vector<double> GAAnalysis::avgFunctionFitness
```

List of the average fitness per FunctionData structure.

3.7.2.2 executionTimes

```
vector<double> GAAnalysis::executionTimes
```

List of execution times in ms for all functions.

3.7.2.3 functionCalls

```
vector<int> GAAnalysis::functionCalls
```

List of the amount of times a function was called.

3.7.2.4 functionIDs

`vector<int> GAAnalysis::functionIDs`

List of function IDs.

3.7.2.5 header

```
string GAAnalysis::header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time(
Calls\n"
```

Header used when saving the data.

3.7.2.6 medianFunctionFitness

`vector<double> GAAnalysis::medianFunctionFitness`

List of the Median fitness from each FunctionData structure.

3.7.2.7 ranges

`vector<vector<double> > GAAnalysis::ranges`

List of ranges for each fitness result in resultsOfFunctions.

3.7.2.8 standardDeviation

`vector<double> GAAnalysis::standardDeviation`

List of standard fitness deviations.

The documentation for this struct was generated from the following file:

- C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/GeneticAlgorithm.h

3.8 GeneticAlgorithm Class Reference

Public Member Functions

- GeneticAlgorithm (int dim, int sol, int gen, double cr, double mProb, double mR, double mPrec, double er, int selectionID, int crPoints)
  The only constructor available to initialize the Genetic Algorithm.
- void setPopulationParams (int functionID, vector< double > bounds)
  Sets up the population parameters.
- double runGeneticAlgorithm ()
  Runs the Genetic Algorithm with set parameters.
- void analyzeGAResults ()
  Analyzes the results of the Genetic Algorithm.
- void printGAResults ()
  Prints the Results of the Genetic Algorithm.
- void printGAAnalysis ()
  Prints the Analysis of the Genetic Algorithm.
- void saveGAResults (string iniFilename)
  Saves all Genetic Algorithm Results to file.
- void saveGAAnalysis ()
  Saves the Analysis of the Genetic Algorithm to file.

Private Member Functions

- void generateRandPopulation ()

  Generates a random initial population for the Genetic Algorithm.
- void evaluatePopulation (int functionID, vector< vector< double >> &pop, vector< double > &fitness, int &functionCounter)

  Calculates fitness of all solutions in population.
- vector< int > select (int selectionID, vector< double > &popFitness, mt19937 &randGenerator)

  Selects two parents from the population.
- int t_select (vector< double > &popFitness, int numOfInid, mt19937 &randGenerator)

  Uses Tournament Selection to select one parent from population.
- int rw_select (vector< double > &popFitness, mt19937 &randGenerator)

  Uses Roulette Wheel Selection to select one parent from population.
- vector< vector< double > > crossover (int crPoints, vector< double > parent1, vector< double > parent2, double cr, mt19937 &randGenerator)

  Creates two children using the two parent individuals.
- vector< vector< double > > crossover1 (vector< double > parent1, vector< double > parent2, double cr, mt19937 &randGenerator)

  Returns two children that are a crossover of the parents using 1 crossover point.
- vector< vector< double > > crossover2 (vector< double > parent1, vector< double > parent2, double cr, mt19937 &randGenerator)

  Returns two or six children that are a crossover of the parents using 2 crossover points.
- void mutate (vector< double > &child, double mProb, double mRange, double mPrec, vector< double > bounds, mt19937 &randGenerator)

  Mutates the child.
- void reduce (GA_Population &pop, vector< vector< double >> &newPop, vector< double > &newFit, int eliteIndex)

  Combines new population with the old one.
- void normalizeFitness (int functionID, vector< double > &popFitness)

  Normalizes all fitness values for a population.
- void recordBestFitness (GA_Population &pop)

  Saves the best fitness value from the population into the GA_Population population structure.

Private Attributes

- GA_Config **gaConfig**
- vector< GA_Population > **popList**
- GA_Population **population**
- GAAnalysis **gaAnalysis**

3.8.1 Constructor & Destructor Documentation

3.8.1.1 GeneticAlgorithm()

```
GeneticAlgorithm::GeneticAlgorithm (
            int dim,
            int sol,
            int gen,
            double cr,
            double mProb,
            double mR,
            double mPrec,
            double er,
            int selectionID,
            int crPoints )
```

The only constructor available to initialize the Genetic Algorithm.

Note

No Default (zero-param) Constructor exists.

Parameters

dim	The number of dimensions each individual in the population has.
sol	The population size.
gen	The max number of generations possible.
cr	The crossover probability.
mProb	The mutation probability.
mR	The mutation range.
mPrec	The mutation precision.
er	The elitism rate.
selectionID	The selection type to use for the Genetic Algorithm.
crPoints	The number of crossover points.

3.8.2 Member Function Documentation

3.8.2.1 analyzeGAResults()

```
void GeneticAlgorithm::analyzeGAResults ( )
```

Analyzes the results of the Genetic Algorithm.

Analyzes the results of the Genetic Algorithm.

3.8.2.2 crossover()

```
vector< vector< double > > GeneticAlgorithm::crossover (
            int crPoints,
            vector< double > parent1,
            vector< double > parent2,
            double cr,
            mt19937 & randGenerator ) [private]
```

Creates two children using the two parent individuals.

Returns two children that are a crossover of the parents.

Parameters

crPoints	The number of crossover points.
parent1	The first parent.
parent2	The second parent.
cr	The crossover probability.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

Returns two children that are a crossover of the parents.

3.8.2.3 crossover1()

```
vector< vector< double > > GeneticAlgorithm::crossover1 (
            vector< double > parent1,
            vector< double > parent2,
            double cr,
            mt19937 & randGenerator ) [private]
```

Returns two children that are a crossover of the parents using 1 crossover point.

1 Crossover - Returns two children that are a crossover of the parents.

Parameters

parent1	The first parent.
parent2	The second parent.
cr	The crossover probability.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

Returns two children that are a crossover of the parents.

3.8.2.4 crossover2()

```
vector< vector< double > > GeneticAlgorithm::crossover2 (
            vector< double > parent1,
            vector< double > parent2,
            double cr,
            mt19937 & randGenerator ) [private]
```

Returns two or six children that are a crossover of the parents using 2 crossover points.

2 Crossovers - Returns two children that are a crossover of the parents.

Parameters

parent1	The first parent.
parent2	The second parent.
cr	The crossover probability.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

Returns two or six children that are a crossover of the parents.

3.8.2.5 evaluatePopulation()

```
void GeneticAlgorithm::evaluatePopulation (
            int functionID,
            vector< vector< double >> & pop,
            vector< double > & fitness,
            int & functionCounter ) [private]
```

Calculates fitness of all solutions in population.

Calculates fitness of all solutions in population.

Note

Makes function call to EA_Utilities.h --> calculateFitnessOfVector().

Parameters

functionID	The ID of the benchmark function to use.
pop	The matrix population of the Genetic Algorithm.
fitness	The fitness vector for each solution from the population.
functionCounter	A counter to keep track of how many times fitness function was called.

3.8.2.6 generateRandPopulation()

```
void GeneticAlgorithm::generateRandPopulation ( )  [private]
```

Generates a random initial population for the Genetic Algorithm.

Generates the initial population for Genetic Algorithm.

Note

Makes function call to EA_Utilities.h --> createMatrix().

3.8.2.7 mutate()

```
void GeneticAlgorithm::mutate (
            vector< double > & child,
            double mProb,
            double mRange,
            double mPrec,
            vector< double > bounds,
            mt19937 & randGenerator )  [private]
```

Mutates the child.

Mutates the child.

Parameters

child	The child vector to be mutated.
mProb	Mutation Probability.
mRange	Mutation Range.
mPrec	Mutation Precision.
bounds	The min/max bounds of the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

3.8.2.8 normalizeFitness()

```
void GeneticAlgorithm::normalizeFitness (
            int functionID,
            vector< double > & popFitness )  [private]
```

Normalizes all fitness values for a population.

Normalizes all fitness values for a population.

Parameters

functionID	The ID of the benchmark function to normalize to.
popFitness	A vector of fitness values from the population.

3.8.2.9 printGAAnalysis()

```
void GeneticAlgorithm::printGAAnalysis ( )
```

Prints the Analysis of the Genetic Algorithm.

Prints the Analysis of the Genetic Algorithm.

3.8.2.10 printGAResults()

```
void GeneticAlgorithm::printGAResults ( )
```

Prints the Results of the Genetic Algorithm.

Prints the Results of the Genetic Algorithm.

3.8.2.11 recordBestFitness()

```
void GeneticAlgorithm::recordBestFitness (
            GA_Population & pop )  [private]
```

Saves the best fitness value from the population into the GA_Population population structure.

Saves the best fitness value from the population.

Note

Makes function call to EA_Utilities.h --> quicksort().

Parameters

pop	The GA_Population population structure.

3.8.2.12 reduce()

```
void GeneticAlgorithm::reduce (
            GA_Population & pop,
```

```
            vector< vector< double >> & newPop,
            vector< double > & newFit,
            int eliteIndex )  [private]
```

Combines new population with the old one.

Combines new population with the old one.

Note

Makes function call to EA_Utilities.h --> quicksort().

Parameters

pop	The GA_Population population structure.
newPop	The new population.
newFit	The new set of fitness values for the new population.
eliteIndex	The ending index of how much of the old population to rertain.

3.8.2.13 runGeneticAlgorithm()

```
double GeneticAlgorithm::runGeneticAlgorithm ( )
```

Runs the Genetic Algorithm with set parameters.

Runs the Genetic Algorithm with set parameters.

Returns

Returns the best result of the Genetic Algorithm.

3.8.2.14 rw_select()

```
int GeneticAlgorithm::rw_select (
            vector< double > & popFitness,
            mt19937 & randGenerator )  [private]
```

Uses Roulette Wheel Selection to select one parent from population.

Uses Roulette Wheel Selection to select one parent from population.

Note

All fitness values have been normalized to between 0 to 1.

Parameters

popFitness	The fitness list of the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

The index of the parent in the population.

3.8.2.15 saveGAAnalysis()

```
void GeneticAlgorithm::saveGAAnalysis ( )
```

Saves the Analysis of the Genetic Algorithm to file.

Saves the Analysis of the Genetic Algorithm to file.

3.8.2.16 saveGAResults()

```
void GeneticAlgorithm::saveGAResults (
             string iniFilename )
```

Saves all Genetic Algorithm Results to file.

Saves all Genetic Algorithm Results to file.

Parameters

iniFilename	The name of the initialization file that was used to set up the population parameters of the Genetic Algorithm.

3.8.2.17 select()

```
vector< int > GeneticAlgorithm::select (
             int selectionID,
             vector< double > & popFitness,
             mt19937 & randGenerator )  [private]
```

Selects two parents from the population.

Selects two parents from the population.

Note

Makes function call to EA_Utilities.h --> printAllGASelectionIDs().

Parameters

selectionID	The ID of the selection type to use.
popFitness	The fitness list of the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A list of parent indices.

3.8.2.18 setPopulationParams()

```
void GeneticAlgorithm::setPopulationParams (
            int functionID,
            vector< double > bounds )
```

Sets up the population parameters.

Sets up the parameters for the population.

Note

The bounds vector list must have min bound at index 0 and max bound at index 1.

Parameters

functionID	The ID of which benchmark function to use (IDs: 1 - 18).
bounds	A pair of min/max boundaries for the individuals in the population.

3.8.2.19 t_select()

```
int GeneticAlgorithm::t_select (
            vector< double > & popFitness,
            int numOfInid,
            mt19937 & randGenerator ) [private]
```

Uses Tournament Selection to select one parent from population.

Uses Tournament Selection to select one parent from population.

Note

All fitness values have been normalized to between 0 to 1.

Parameters

popFitness	The fitness list of the population.
numOfInid	The number of individuals to select for tournament (max = size of population).
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

The index of the parent in the population.

The documentation for this class was generated from the following files:

- C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/GeneticAlgorithm.h
- C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/GeneticAlgorithm.cpp

Chapter 4

File Documentation

4.1 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/BenchmarkFunctions.cpp File Reference

A library of benchmark functions.

```
#include "BenchmarkFunctions.h"
```

Functions

- double schefelsFunc (vector< double > &vect, int size)

  Performs the Schefel's Function on a vector of elements.
- double deJongsFunc (vector< double > &vect, int size)

  Performs the 1st De Jong's Function on a vector of elements.
- double rosenbrockFunc (vector< double > &vect, int size)

  Performs the Rosenbrock Function on a vector of elements.
- double rastriginFunc (vector< double > &vect, int size)

  Performs the Rastrigin Function on a vector of elements.
- double griewangkFunc (vector< double > &vect, int size)

  Performs the Griewangk Function on a vector of elements.
- double sineEnvelopeSineWaveFunc (vector< double > &vect, int size)

  Performs the Sine Envelope Sine Wave Function on a vector of elements.
- double stretchedVSineWaveFunc (vector< double > &vect, int size)

  Performs the Stretched V Sine Wave Function on a vector of elements.
- double ackleysOneFunc (vector< double > &vect, int size)

  Performs the Ackley's One Function on a vector of elements.
- double ackleysTwoFunc (vector< double > &vect, int size)

  Performs the Ackley's Two Function on a vector of elements.
- double eggHolderFunc (vector< double > &vect, int size)

  Performs the Egg Holder Function on a vector of elements.
- double ranaFunc (vector< double > &vect, int size)

  Performs the Rana Function on a vector of elements.
- double pathologicalFunc (vector< double > &vect, int size)

  Performs the Pathological Function on a vector of elements.

- double michalewiczFunc (vector< double > &vect, int size)

    Performs the Michalewicz Function on a vector of elements.
- double mastersCosWaveFunc (vector< double > &vect, int size)

    Performs the Masters Cosine Wave Function on a vector of elements.
- double quarticFunc (vector< double > &vect, int size)

    Performs the Quartic Function on a vector of elements.
- double levyFunc (vector< double > &vect, int size)

    Performs the Levy Function on a vector of elements.
- double stepFunc (vector< double > &vect, int size)

    Performs the Step Function on a vector of elements.
- double alpineFunc (vector< double > &vect, int size)

    Performs the Alpine Function on a vector of elements.

4.1.1 Detailed Description

A library of benchmark functions.

Author

Al Timofeyev

Date

April 17, 2019

4.1.2 Function Documentation

4.1.2.1 ackleysOneFunc()

```
double ackleysOneFunc (
            vector< double > & vect,
            int size )
```

Performs the Ackley's One Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.2 ackleysTwoFunc()

```
double ackleysTwoFunc (
            vector< double > & vect,
            int size )
```

Performs the Ackley's Two Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.3 alpineFunc()

```
double alpineFunc (
            vector< double > & vect,
            int size )
```

Performs the Alpine Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.4 deJongsFunc()

```
double deJongsFunc (
            vector< double > & vect,
            int size )
```

Performs the 1st De Jong's Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.5 eggHolderFunc()

```
double eggHolderFunc (
            vector< double > & vect,
            int size )
```

Performs the Egg Holder Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.6 griewangkFunc()

```
double griewangkFunc (
            vector< double > & vect,
            int size )
```

Performs the Griewangk Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.7 levyFunc()

```
double levyFunc (
            vector< double > & vect,
            int size )
```

Performs the Levy Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.8 mastersCosWaveFunc()

```
double mastersCosWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Masters Cosine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.9 michalewiczFunc()

```
double michalewiczFunc (
            vector< double > & vect,
            int size )
```

Performs the Michalewicz Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.10 pathologicalFunc()

```
double pathologicalFunc (
            vector< double > & vect,
            int size )
```

Performs the Pathological Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.11 quarticFunc()

```
double quarticFunc (
            vector< double > & vect,
            int size )
```

Performs the Quartic Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.12 ranaFunc()

```
double ranaFunc (
            vector< double > & vect,
            int size )
```

Performs the Rana Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.13 rastriginFunc()

```
double rastriginFunc (
            vector< double > & vect,
            int size )
```

Performs the Rastrigin Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.14 rosenbrockFunc()

```
double rosenbrockFunc (
            vector< double > & vect,
            int size )
```

Performs the Rosenbrock Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.15 schefelsFunc()

```
double schefelsFunc (
            vector< double > & vect,
            int size )
```

Performs the Schefel's Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.16 sineEnvelopeSineWaveFunc()

```
double sineEnvelopeSineWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Sine Envelope Sine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.17 stepFunc()

```
double stepFunc (
            vector< double > & vect,
            int size )
```

Performs the Step Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.18 stretchedVSineWaveFunc()

```
double stretchedVSineWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Stretched V Sine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/Benchmark↩ Functions.h File Reference

A library of benchmark functions.

```
#include <vector>
#include <math.h>
#include <cmath>
```

Functions

- double schefelsFunc (vector< double > &vect, int size)

  Performs the Schefel's Function on a vector of elements.
- double deJongsFunc (vector< double > &vect, int size)

  Performs the 1st De Jong's Function on a vector of elements.
- double rosenbrockFunc (vector< double > &vect, int size)

  Performs the Rosenbrock Function on a vector of elements.
- double rastriginFunc (vector< double > &vect, int size)

  Performs the Rastrigin Function on a vector of elements.
- double griewangkFunc (vector< double > &vect, int size)

  Performs the Griewangk Function on a vector of elements.
- double sineEnvelopeSineWaveFunc (vector< double > &vect, int size)

  Performs the Sine Envelope Sine Wave Function on a vector of elements.
- double stretchedVSineWaveFunc (vector< double > &vect, int size)

  Performs the Stretched V Sine Wave Function on a vector of elements.
- double ackleysOneFunc (vector< double > &vect, int size)

  Performs the Ackley's One Function on a vector of elements.
- double ackleysTwoFunc (vector< double > &vect, int size)

  Performs the Ackley's Two Function on a vector of elements.
- double eggHolderFunc (vector< double > &vect, int size)

  Performs the Egg Holder Function on a vector of elements.

- double ranaFunc (vector< double > &vect, int size)

    Performs the Rana Function on a vector of elements.
- double pathologicalFunc (vector< double > &vect, int size)

    Performs the Pathological Function on a vector of elements.
- double michalewiczFunc (vector< double > &vect, int size)

    Performs the Michalewicz Function on a vector of elements.
- double mastersCosWaveFunc (vector< double > &vect, int size)

    Performs the Masters Cosine Wave Function on a vector of elements.
- double quarticFunc (vector< double > &vect, int size)

    Performs the Quartic Function on a vector of elements.
- double levyFunc (vector< double > &vect, int size)

    Performs the Levy Function on a vector of elements.
- double stepFunc (vector< double > &vect, int size)

    Performs the Step Function on a vector of elements.
- double alpineFunc (vector< double > &vect, int size)

    Performs the Alpine Function on a vector of elements.

4.2.1 Detailed Description

A library of benchmark functions.

Author

Al Timofeyev

Date

April 17, 2019

4.2.2 Function Documentation

4.2.2.1 ackleysOneFunc()

```
double ackleysOneFunc (
            vector< double > & vect,
            int size )
```

Performs the Ackley's One Function on a vector of elements.

Performs the Ackley's One Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.2 ackleysTwoFunc()

```
double ackleysTwoFunc (
            vector< double > & vect,
            int size )
```

Performs the Ackley's Two Function on a vector of elements.

Performs the Ackley's Two Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.3 alpineFunc()

```
double alpineFunc (
            vector< double > & vect,
            int size )
```

Performs the Alpine Function on a vector of elements.

Performs the Alpine Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.4 deJongsFunc()

```
double deJongsFunc (
            vector< double > & vect,
            int size )
```

Performs the 1st De Jong's Function on a vector of elements.

Performs the 1st De Jong's Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.5 eggHolderFunc()

```
double eggHolderFunc (
            vector< double > & vect,
            int size )
```

Performs the Egg Holder Function on a vector of elements.

Performs the Egg Holder Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.6 griewangkFunc()

```
double griewangkFunc (
            vector< double > & vect,
            int size )
```

Performs the Griewangk Function on a vector of elements.

Performs the Griewangk Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.7 levyFunc()

```
double levyFunc (
            vector< double > & vect,
            int size )
```

Performs the Levy Function on a vector of elements.

Performs the Levy Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.8 mastersCosWaveFunc()

```
double mastersCosWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Masters Cosine Wave Function on a vector of elements.

Performs the Masters Cosine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.9 michalewiczFunc()

```
double michalewiczFunc (
            vector< double > & vect,
            int size )
```

Performs the Michalewicz Function on a vector of elements.

Performs the Michalewicz Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.10 pathologicalFunc()

```
double pathologicalFunc (
            vector< double > & vect,
            int size )
```

Performs the Pathological Function on a vector of elements.

Performs the Pathological Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.11 quarticFunc()

```
double quarticFunc (
            vector< double > & vect,
            int size )
```

Performs the Quartic Function on a vector of elements.

Performs the Quartic Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.12 ranaFunc()

```
double ranaFunc (
            vector< double > & vect,
            int size )
```

Performs the Rana Function on a vector of elements.

Performs the Rana Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.13 rastriginFunc()

```
double rastriginFunc (
            vector< double > & vect,
            int size )
```

Performs the Rastrigin Function on a vector of elements.

Performs the Rastrigin Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.14 rosenbrockFunc()

```
double rosenbrockFunc (
            vector< double > & vect,
            int size )
```

Performs the Rosenbrock Function on a vector of elements.

Performs the Rosenbrock Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.15 schefelsFunc()

```
double schefelsFunc (
            vector< double > & vect,
            int size )
```

Performs the Schefel's Function on a vector of elements.

Performs the Schefel's Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.16 sineEnvelopeSineWaveFunc()

```
double sineEnvelopeSineWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Sine Envelope Sine Wave Function on a vector of elements.

Performs the Sine Envelope Sine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.17 stepFunc()

```
double stepFunc (
            vector< double > & vect,
            int size )
```

Performs the Step Function on a vector of elements.

Performs the Step Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.18 stretchedVSineWaveFunc()

```
double stretchedVSineWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Stretched V Sine Wave Function on a vector of elements.

Performs the Stretched V Sine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.3 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/DEA_↩ Strategies.cpp File Reference

A library of strategies used in the Differential Evolution Algorithm.

```
#include "DEA_Strategies.h"
```

Functions

- vector< double > de_Strategy1 (const double &cr, const double &F, vector< double > x, vector< double > xBest, vector< double > xRand2, vector< double > xRand3, mt19937 &randGenerator)

  Strategy 1: DE/best/1/exp.
- vector< double > de_Strategy2 (const double &cr, const double &F, vector< double > x, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, mt19937 &randGenerator)

  Strategy 2: DE/rand/1/exp.
- vector< double > de_Strategy3 (const double &cr, const double &F, const double &lambda, vector< double > x, vector< double > xBest, vector< double > xRand1, vector< double > xRand2, mt19937 &rand↩ Generator)

  Strategy 3: DE/rand-to-best/1/exp.
- vector< double > de_Strategy4 (const double &cr, const double &F, vector< double > x, vector< double > xBest, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, vector< double > xRand4, mt19937 &randGenerator)

  Strategy 4: DE/best/2/exp.
- vector< double > de_Strategy5 (const double &cr, const double &F, vector< double > x, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, vector< double > xRand4, vector< double > xRand5, mt19937 &randGenerator)

  Strategy 5: DE/rand/2/exp.
- vector< double > de_Strategy6 (const double &cr, const double &F, vector< double > x, vector< double > xBest, vector< double > xRand2, vector< double > xRand3, mt19937 &randGenerator)

  Strategy 6: DE/best/1/bin.

- vector< double > de_Strategy7 (const double &cr, const double &F, vector< double > x, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, mt19937 &randGenerator)

  Strategy 7: DE/rand/1/bin.
- vector< double > de_Strategy8 (const double &cr, const double &F, const double &lambda, vector< double > x, vector< double > xBest, vector< double > xRand1, vector< double > xRand2, mt19937 &rand↩Generator)

  Strategy 8: DE/rand-to-best/1/bin.
- vector< double > de_Strategy9 (const double &cr, const double &F, vector< double > x, vector< double > xBest, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, vector< double > xRand4, mt19937 &randGenerator)

  Strategy 9: DE/best/2/bin.
- vector< double > de_Strategy10 (const double &cr, const double &F, vector< double > x, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, vector< double > xRand4, vector< double > xRand5, mt19937 &randGenerator)

  Strategy 10: DE/rand/2/bin.

4.3.1 Detailed Description

A library of strategies used in the Differential Evolution Algorithm.

Author

Al Timofeyev

Date

May 2, 2019 The general notation used for these strategies is DE/x/y/z: where DE stands for Differential Evolution algorithm, x represents a string denoting the vector to be perturbed, y is the number of difference vectors considered for perturbation of x, and z is the type of crossover being used. Two types of crossovers: exp (exponential) and bin (binomial).

4.3.2 Function Documentation

4.3.2.1 de_Strategy1()

```
vector<double> de_Strategy1 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xBest,
            vector< double > xRand2,
            vector< double > xRand3,
            mt19937 & randGenerator )
```

Strategy 1: DE/best/1/exp.

Note

xBest != xRand2 != xRand3 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.3.2.2 de_Strategy10()

```
vector<double> de_Strategy10 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xRand1,
            vector< double > xRand2,
            vector< double > xRand3,
            vector< double > xRand4,
            vector< double > xRand5,
            mt19937 & randGenerator )
```

Strategy 10: DE/rand/2/bin.

Note

xRand1 != xRand2 != xRand3 != xRand4 != xRand5 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
xRand4	A randomly chosen solution from the population.
xRand5	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.3.2.3 de_Strategy2()

```
vector<double> de_Strategy2 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xRand1,
            vector< double > xRand2,
            vector< double > xRand3,
            mt19937 & randGenerator )
```

Strategy 2: DE/rand/1/exp.

Note

xRand1 != xRand2 != xRand3 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.3.2.4 de_Strategy3()

```
vector<double> de_Strategy3 (
            const double & cr,
            const double & F,
            const double & lambda,
            vector< double > x,
            vector< double > xBest,
            vector< double > xRand1,
            vector< double > xRand2,
            mt19937 & randGenerator )
```

Strategy 3: DE/rand-to-best/1/exp.

Note

xBest != xRand1 != xRand2 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
lambda	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.3.2.5 de_Strategy4()

```
vector<double> de_Strategy4 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xBest,
            vector< double > xRand1,
            vector< double > xRand2,
            vector< double > xRand3,
            vector< double > xRand4,
            mt19937 & randGenerator )
```

Strategy 4: DE/best/2/exp.

Note

xBest != xRand1 != xRand2 != xRand3 != xRand4 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
xRand4	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.3.2.6 de_Strategy5()

```
vector<double> de_Strategy5 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xRand1,
            vector< double > xRand2,
            vector< double > xRand3,
            vector< double > xRand4,
            vector< double > xRand5,
            mt19937 & randGenerator )
```

Strategy 5: DE/rand/2/exp.

Note

xRand1 != xRand2 != xRand3 != xRand4 != xRand5 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
xRand4	A randomly chosen solution from the population.
xRand5	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.3.2.7 de_Strategy6()

```
vector<double> de_Strategy6 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xBest,
```

```
            vector< double > xRand2,
            vector< double > xRand3,
            mt19937 & randGenerator )
```

Strategy 6: DE/best/1/bin.

Note

xBest != xRand2 != xRand3 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.3.2.8 de_Strategy7()

```
vector<double> de_Strategy7 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xRand1,
            vector< double > xRand2,
            vector< double > xRand3,
            mt19937 & randGenerator )
```

Strategy 7: DE/rand/1/bin.

Note

xRand1 != xRand2 != xRand3 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.3.2.9 de_Strategy8()

```
vector<double> de_Strategy8 (
            const double & cr,
            const double & F,
            const double & lambda,
            vector< double > x,
            vector< double > xBest,
            vector< double > xRand1,
            vector< double > xRand2,
            mt19937 & randGenerator )
```

Strategy 8: DE/rand-to-best/1/bin.

Note

xBest != xRand1 != xRand2 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
lambda	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.3.2.10 de_Strategy9()

```
vector<double> de_Strategy9 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xBest,
            vector< double > xRand1,
            vector< double > xRand2,
```

```
vector< double > xRand3,
vector< double > xRand4,
mt19937 & randGenerator )
```

Strategy 9: DE/best/2/bin.

Note

xBest != xRand1 != xRand2 != xRand3 != xRand4 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
xRand4	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.4 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/DEA_Strategies.h File Reference

A library of strategies used in the Differential Evolution Algorithm.

```
#include <vector>
#include <random>
```

Functions

- vector< double > de_Strategy1 (const double &cr, const double &F, vector< double > x, vector< double > xBest, vector< double > xRand2, vector< double > xRand3, mt19937 &randGenerator)

  Strategy 1: DE/best/1/exp.
- vector< double > de_Strategy2 (const double &cr, const double &F, vector< double > x, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, mt19937 &randGenerator)

  Strategy 2: DE/rand/1/exp.
- vector< double > de_Strategy3 (const double &cr, const double &F, const double &lambda, vector< double > x, vector< double > xBest, vector< double > xRand1, vector< double > xRand2, mt19937 &randGenerator)

  Strategy 3: DE/rand-to-best/1/exp.

- vector< double > de_Strategy4 (const double &cr, const double &F, vector< double > x, vector< double > xBest, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, vector< double > xRand4, mt19937 &randGenerator)

  Strategy 4: DE/best/2/exp.

- vector< double > de_Strategy5 (const double &cr, const double &F, vector< double > x, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, vector< double > xRand4, vector< double > xRand5, mt19937 &randGenerator)

  Strategy 5: DE/rand/2/exp.

- vector< double > de_Strategy6 (const double &cr, const double &F, vector< double > x, vector< double > xBest, vector< double > xRand2, vector< double > xRand3, mt19937 &randGenerator)

  Strategy 6: DE/best/1/bin.

- vector< double > de_Strategy7 (const double &cr, const double &F, vector< double > x, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, mt19937 &randGenerator)

  Strategy 7: DE/rand/1/bin.

- vector< double > de_Strategy8 (const double &cr, const double &F, const double &lambda, vector< double > x, vector< double > xBest, vector< double > xRand1, vector< double > xRand2, mt19937 &randGenerator)

  Strategy 8: DE/rand-to-best/1/bin.

- vector< double > de_Strategy9 (const double &cr, const double &F, vector< double > x, vector< double > xBest, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, vector< double > xRand4, mt19937 &randGenerator)

  Strategy 9: DE/best/2/bin.

- vector< double > de_Strategy10 (const double &cr, const double &F, vector< double > x, vector< double > xRand1, vector< double > xRand2, vector< double > xRand3, vector< double > xRand4, vector< double > xRand5, mt19937 &randGenerator)

  Strategy 10: DE/rand/2/bin.

4.4.1 Detailed Description

A library of strategies used in the Differential Evolution Algorithm.

Author

Al Timofeyev

Date

May 2, 2019 The general notation used for these strategies is DE/x/y/z: where DE stands for Differential Evolution algorithm, x represents a string denoting the vector to be perturbed, y is the number of difference vectors considered for perturbation of x, and z is the type of crossover being used. Two types of crossovers: exp (exponential) and bin (binomial).

4.4.2 Function Documentation

4.4.2.1 de_Strategy1()

```
vector<double> de_Strategy1 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xBest,
            vector< double > xRand2,
            vector< double > xRand3,
            mt19937 & randGenerator )
```

Strategy 1: DE/best/1/exp.

Strategy 1: DE/best/1/exp

Note

xBest != xRand2 != xRand3 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.4.2.2 de_Strategy10()

```
vector<double> de_Strategy10 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xRand1,
            vector< double > xRand2,
            vector< double > xRand3,
            vector< double > xRand4,
            vector< double > xRand5,
            mt19937 & randGenerator )
```

Strategy 10: DE/rand/2/bin.

Strategy 10: DE/rand/2/bin

Note

xRand1 != xRand2 != xRand3 != xRand4 != xRand5 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
xRand4	A randomly chosen solution from the population.
xRand5	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.4.2.3 de_Strategy2()

```
vector<double> de_Strategy2 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xRand1,
            vector< double > xRand2,
            vector< double > xRand3,
            mt19937 & randGenerator )
```

Strategy 2: DE/rand/1/exp.

Strategy 2: DE/rand/1/exp

Note

xRand1 != xRand2 != xRand3 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.4.2.4 de_Strategy3()

```
vector<double> de_Strategy3 (
            const double & cr,
            const double & F,
            const double & lambda,
            vector< double > x,
            vector< double > xBest,
            vector< double > xRand1,
            vector< double > xRand2,
            mt19937 & randGenerator )
```

Strategy 3: DE/rand-to-best/1/exp.

Strategy 3: DE/rand-to-best/1/exp

Note

xBest != xRand1 != xRand2 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
lambda	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.4.2.5 de_Strategy4()

```
vector<double> de_Strategy4 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xBest,
```

```
vector< double > xRand1,
vector< double > xRand2,
vector< double > xRand3,
vector< double > xRand4,
mt19937 & randGenerator )
```

Strategy 4: DE/best/2/exp.

Strategy 4: DE/best/2/exp

Note

xBest != xRand1 != xRand2 != xRand3 != xRand4 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
xRand4	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.4.2.6 de_Strategy5()

```
vector<double> de_Strategy5 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xRand1,
            vector< double > xRand2,
            vector< double > xRand3,
            vector< double > xRand4,
            vector< double > xRand5,
            mt19937 & randGenerator )
```

Strategy 5: DE/rand/2/exp.

Strategy 5: DE/rand/2/exp

Note

xRand1 != xRand2 != xRand3 != xRand4 != xRand5 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
xRand4	A randomly chosen solution from the population.
xRand5	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.4.2.7 de_Strategy6()

```
vector<double> de_Strategy6 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xBest,
            vector< double > xRand2,
            vector< double > xRand3,
            mt19937 & randGenerator )
```

Strategy 6: DE/best/1/bin.

Strategy 6: DE/best/1/bin

Note

xBest != xRand2 != xRand3 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.4.2.8 de_Strategy7()

```
vector<double> de_Strategy7 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xRand1,
            vector< double > xRand2,
            vector< double > xRand3,
            mt19937 & randGenerator )
```

Strategy 7: DE/rand/1/bin.

Strategy 7: DE/rand/1/bin

Note

xRand1 != xRand2 != xRand3 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.4.2.9 de_Strategy8()

```
vector<double> de_Strategy8 (
            const double & cr,
            const double & F,
            const double & lambda,
            vector< double > x,
            vector< double > xBest,
            vector< double > xRand1,
```

```
vector< double > xRand2,
mt19937 & randGenerator )
```

Strategy 8: DE/rand-to-best/1/bin.

Strategy 8: DE/rand-to-best/1/bin

Note

xBest != xRand1 != xRand2 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
lambda	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.4.2.10 de_Strategy9()

```
vector<double> de_Strategy9 (
            const double & cr,
            const double & F,
            vector< double > x,
            vector< double > xBest,
            vector< double > xRand1,
            vector< double > xRand2,
            vector< double > xRand3,
            vector< double > xRand4,
            mt19937 & randGenerator )
```

Strategy 9: DE/best/2/bin.

Strategy 9: DE/best/2/bin

Note

xBest != xRand1 != xRand2 != xRand3 != xRand4 (where "!=" means "not equal to").

Parameters

cr	The crossover probability.
F	A scaling factor.
x	The current solution (individual) of the population.
xBest	The best solution of the population.
xRand1	A randomly chosen solution from the population.
xRand2	A randomly chosen solution from the population.
xRand3	A randomly chosen solution from the population.
xRand4	A randomly chosen solution from the population.
randGenerator	A Mersenne Twister pseudo-random number generator.

Returns

A new solution (individual).

4.5 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/DifferentialEvolution.cpp File Reference

Implementation of the Differential Evolution Algorithm.

```
#include "DifferentialEvolution.h"
```

4.5.1 Detailed Description

Implementation of the Differential Evolution Algorithm.

Author

Al Timofeyev

Date

May 3, 2019

4.6 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/DifferentialEvolution.h File Reference

Implementation of the Differential Evolution Algorithm.

```
#include <fstream>
#include <chrono>
#include "DEA_Strategies.h"
#include "EA_Utilities.h"
```

Classes

- struct DEA_Config

  Holds all the user defined variables. Differential Evolution Algorithm Configuration Structure, where all user defined variables that are used to configure the Differential Evolution Algorithm are stored.
- struct DEA_Population

  Holds all the population information. Differential Evolution Algorithm Population Structure, holds all the data related to the population of the Differential Evolution Algorithm.
- struct DEAAnalysis

  Differential Evolution Algorithm Analysis Differential Evolution Algorithm Analysis Structure, to keep track of the analysis performed on each population in the population list.
- class DifferentialEvolution

4.6.1 Detailed Description

Implementation of the Differential Evolution Algorithm.

Author

Al Timofeyev

Date

May 3, 2019

4.7 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/EA_↩ Utilities.cpp File Reference

This utilities file is used by the Evolutionary Algorithms, and to create matrices using the Mersenne Twister.

```
#include "EA_Utilities.h"
```

Functions

- void printAllFunctionIDs ()

  Prints all the possible Function IDs to the screen.
- void printAllGASelectionIDs ()

  Prints all the possible Genetic Algorithm Selection IDs to the screen.
- vector< vector< double > > createGAMatrix (int rows, int columns, double minBound, double maxBound)

  Creates a matrix of doubles using Mersenne Twister for the Genetic Algorithm.
- vector< vector< double > > createDEAMatrix (int rows, int columns, double minBound, double maxBound)

  Creates a matrix of doubles using Mersenne Twister for the Differential Evolution Algorithm.
- double calculateFitnessOfVector (vector< double > &vect, int functionID)

  Calculates the fitness of a vector.
- vector< double > calculateFitnessOfMatrix (vector< vector< double >> matrix, int functionID)

  Calculates the fitness of all vectors of a matrix.
- double calculateAverage (vector< double > vect)

Calculates the average value of a vector of doubles.
- double calculateStandardDeviation (vector< double > vect)

  Calculates the standard deviation value of a vector of doubles.
- void quicksort (vector< double > &fitnessList, vector< vector< double >> &matrix, int L, int R)

  Sorts a matrix and its fitness vector based on the fitness.
- void swap (vector< double > &fitnessList, vector< vector< double >> &matrix, int x, int y)

  Swaps the fitness' and their corresponding vectors in the matrix.
- void quicksort (vector< double > &vec, int L, int R)

  A normal Quicksort implementation for vector arrays of doubles.
- void swap (vector< double > &v, int x, int y)

  Swaps two values of a vector array of doubles.

4.7.1 Detailed Description

This utilities file is used by the Evolutionary Algorithms, and to create matrices using the Mersenne Twister.

Author

Al Timofeyev

Date

April 27, 2019

4.7.2 Function Documentation

4.7.2.1 calculateAverage()

```
double calculateAverage (
            vector< double > vect )
```

Calculates the average value of a vector of doubles.

Parameters

vect	The vector of doubles.

Returns

The average value of the vector.

4.7.2.2 calculateFitnessOfMatrix()

```
vector<double> calculateFitnessOfMatrix (
            vector< vector< double >> matrix,
            int functionID )
```

Calculates the fitness of all vectors of a matrix.

Calculates the fitness of all the vectors of the matrix stored All the fitness results are stored in the fitness vector variable.

Parameters

matrix	The matrix that holds all the vectors for calculating the fitness.
functionID	The ID of the function to use for calculating the fitness.

Returns

A vector of fitness values.

4.7.2.3 calculateFitnessOfVector()

```
double calculateFitnessOfVector (
            vector< double > & vect,
            int functionID )
```

Calculates the fitness of a vector.

The fitness of a vector is calculated by the Benchmark Function referenced by the functionID.

Note

This function makes a call to BenchmarkFunctions.h.

Parameters

vect	The vector of elements on which the Benchmark Functions operate.
functionID	The ID that references which Benchmark Function to use.

Returns

The fitness of the vector.

4.7.2.4 calculateStandardDeviation()

```
double calculateStandardDeviation (
            vector< double > vect )
```

Calculates the standard deviation value of a vector of doubles.

Parameters

vect	The vector of doubles.

Returns

The standard deviation value of the vector.

4.7.2.5 createDEAMatrix()

```
vector<vector<double> > createDEAMatrix (
            int rows,
            int columns,
            double minBound,
            double maxBound )
```

Creates a matrix of doubles using Mersenne Twister for the Differential Evolution Algorithm.

A matrix is constructed using the Mersenne Twister in the <random> library with the user-specified min/max boundaries.

Parameters

rows	The number of vectors in the matrix.
columns	The number of elements in each vector of the matrix.
minBound,maxBound	The max/min boundaries are the range in which to generate numbers.

Returns

The fully constructed matrix of doubles.

4.7.2.6 createGAMatrix()

```
vector<vector<double> > createGAMatrix (
            int rows,
            int columns,
            double minBound,
            double maxBound )
```

Creates a matrix of doubles using Mersenne Twister for the Genetic Algorithm.

A matrix is constructed using the Mersenne Twister in the <random> library with the user-specified min/max boundaries.

Parameters

rows	The number of vectors in the matrix.
columns	The number of elements in each vector of the matrix.
minBound,maxBound	The max/min boundaries are the range in which to generate numbers.

Returns

The fully constructed matrix of doubles.

4.7.2.7 printAllFunctionIDs()

```
void printAllFunctionIDs ( )
```

Prints all the possible Function IDs to the screen.

Prints all possible Function ID, as well as the functions they reference, to the screen.

4.7.2.8 printAllGASelectionIDs()

```
void printAllGASelectionIDs ( )
```

Prints all the possible Genetic Algorithm Selection IDs to the screen.

Prints all the possible GA Selection IDs to the screen.

4.7.2.9 quicksort() [1/2]

```
void quicksort (
            vector< double > & fitnessList,
            vector< vector< double >> & matrix,
            int L,
            int R )
```

Sorts a matrix and its fitness vector based on the fitness.

Note

Sorted in Ascending Order.
Smallest (minimum) fitness gets moved to index 0, along with its vector from matrix.
Largest (maximum) fitness gets moved to the last index, along with its vector from matrix.

Parameters

fitnessList	The list of fitness values that correspond to each row of the matrix.
matrix	A matrix of double values.
L	The starting index for the quicksort (inclusive).
R	The ending index for the quicksort (inclusive).

4.7.2.10 quicksort() [2/2]

```
void quicksort (
            vector< double > & vec,
            int L,
            int R )
```

A normal Quicksort implementation for vector arrays of doubles.

Note

Sorted in Ascending Order.
Smallest value gets moved to index 0.
Largest value gets moved to the last index.

Parameters

vec	Vector array of doubles.
L	The starting index for the quicksort (inclusive).
R	The ending index for the quicksort (inclusive).

4.7.2.11 swap() [1/2]

```
void swap (
            vector< double > & fitnessList,
            vector< vector< double >> & matrix,
            int x,
            int y )
```

Swaps the fitness' and their corresponding vectors in the matrix.

Parameters

fitnessList	The list of fitness values that correspond to each row of the matrix.
matrix	A matrix of double values.
x	The 1st index of the fitness/vector for the swap.
y	The 2nd index of the fitness/vector for the swap.

4.7.2.12 swap() [2/2]

```
void swap (
            vector< double > & v,
```

```
int x,
int y )
```

Swaps two values of a vector array of doubles.

Parameters

v	The vector in which values are swapped.
x	The 1st index of the fitness/vector for the swap.
y	The 2nd index of the fitness/vector for the swap.

4.8 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/EA_Utilities.h File Reference

This utilities file is used by the Evolutionary Algorithms, and to create matrices using the Mersenne Twister.

```
#include <iostream>
#include <vector>
#include <cmath>
#include <random>
#include "BenchmarkFunctions.h"
```

Functions

- void printAllFunctionIDs ()

  Prints all the possible Function IDs to the screen.
- void printAllGASelectionIDs ()

  Prints all the possible Genetic Algorithm Selection IDs to the screen.
- vector< vector< double > > createGAMatrix (int rows, int columns, double minBound, double maxBound)

  Creates a matrix of doubles using Mersenne Twister for the Genetic Algorithm.
- vector< vector< double > > createDEAMatrix (int rows, int columns, double minBound, double maxBound)

  Creates a matrix of doubles using Mersenne Twister for the Differential Evolution Algorithm.
- double calculateFitnessOfVector (vector< double > &vect, int functionID)

  Calculates the fitness of a vector.
- vector< double > calculateFitnessOfMatrix (vector< vector< double >> matrix, int functionID)

  Calculates the fitness of all vectors of a matrix.
- double calculateAverage (vector< double > vect)

  Calculates the average value of a vector of doubles.
- double calculateStandardDeviation (vector< double > vect)

  Calculates the standard deviation value of a vector of doubles.
- void quicksort (vector< double > &fitnessList, vector< vector< double >> &matrix, int L, int R)

  Sorts a matrix and its fitness vector based on the fitness.
- void swap (vector< double > &fitnessList, vector< vector< double >> &matrix, int x, int y)

  Swaps the fitness' and their corresponding vectors in the matrix.
- void quicksort (vector< double > &vec, int L, int R)

  A normal Quicksort implementation for vector arrays of doubles.
- void swap (vector< double > &v, int x, int y)

  Swaps two values of a vector array of doubles.

4.8.1 Detailed Description

This utilities file is used by the Evolutionary Algorithms, and to create matrices using the Mersenne Twister.

Author

Al Timofeyev

Date

April 27, 2019

4.8.2 Function Documentation

4.8.2.1 calculateAverage()

```
double calculateAverage (
            vector< double > vect )
```

Calculates the average value of a vector of doubles.

Calculates the average value of a vector of doubles.

Parameters

vect	The vector of doubles.

Returns

The average value of the vector.

4.8.2.2 calculateFitnessOfMatrix()

```
vector<double> calculateFitnessOfMatrix (
            vector< vector< double >> matrix,
            int functionID )
```

Calculates the fitness of all vectors of a matrix.

Calculates the fitness of all vectors in matrix.

Calculates the fitness of all the vectors of the matrix stored All the fitness results are stored in the fitness vector variable.

Parameters

matrix	The matrix that holds all the vectors for calculating the fitness.
functionID	The ID of the function to use for calculating the fitness.

Returns

A vector of fitness values.

4.8.2.3 calculateFitnessOfVector()

```
double calculateFitnessOfVector (
            vector< double > & vect,
            int functionID )
```

Calculates the fitness of a vector.

Calculates the fitness of a single vector.

The fitness of a vector is calculated by the Benchmark Function referenced by the functionID.

Note

This function makes a call to BenchmarkFunctions.h.

Parameters

vect	The vector of elements on which the Benchmark Functions operate.
functionID	The ID that references which Benchmark Function to use.

Returns

The fitness of the vector.

4.8.2.4 calculateStandardDeviation()

```
double calculateStandardDeviation (
            vector< double > vect )
```

Calculates the standard deviation value of a vector of doubles.

Calculates the standard deviation value of a vector of doubles.

Parameters

vect	The vector of doubles.

Returns

The standard deviation value of the vector.

4.8.2.5 createDEAMatrix()

```
vector<vector<double> > createDEAMatrix (
            int rows,
            int columns,
            double minBound,
            double maxBound )
```

Creates a matrix of doubles using Mersenne Twister for the Differential Evolution Algorithm.

A matrix is constructed using the Mersenne Twister in the <random> library with the user-specified min/max boundaries.

Parameters

rows	The number of vectors in the matrix.
columns	The number of elements in each vector of the matrix.
minBound,maxBound	The max/min boundaries are the range in which to generate numbers.

Returns

The fully constructed matrix of doubles.

4.8.2.6 createGAMatrix()

```
vector<vector<double> > createGAMatrix (
            int rows,
            int columns,
            double minBound,
            double maxBound )
```

Creates a matrix of doubles using Mersenne Twister for the Genetic Algorithm.

Creates a matrix with the given min/max bound for the given number of rows/columns.

A matrix is constructed using the Mersenne Twister in the <random> library with the user-specified min/max boundaries.

Parameters

rows	The number of vectors in the matrix.
columns	The number of elements in each vector of the matrix.
minBound,maxBound	The max/min boundaries are the range in which to generate numbers.

Returns

The fully constructed matrix of doubles.

4.8.2.7 printAllFunctionIDs()

```
void printAllFunctionIDs ( )
```

Prints all the possible Function IDs to the screen.

Prints all the possible Function IDs to the screen.

Prints all possible Function ID, as well as the functions they reference, to the screen.

4.8.2.8 printAllGASelectionIDs()

```
void printAllGASelectionIDs ( )
```

Prints all the possible Genetic Algorithm Selection IDs to the screen.

Prints all the possible GA Selection IDs to the screen.

4.8.2.9 quicksort() [1/2]

```
void quicksort (
            vector< double > & fitnessList,
            vector< vector< double >> & matrix,
            int L,
            int R )
```

Sorts a matrix and its fitness vector based on the fitness.

Special Quicksort implementation for fitness/matrices.

Note

Sorted in Ascending Order.
Smallest (minimum) fitness gets moved to index 0, along with its vector from matrix.
Largest (maximum) fitness gets moved to the last index, along with its vector from matrix.

Parameters

fitnessList	The list of fitness values that correspond to each row of the matrix.
matrix	A matrix of double values.
L	The starting index for the quicksort (inclusive).
R	The ending index for the quicksort (inclusive).

4.8.2.10 quicksort() [2/2]

```
void quicksort (
            vector< double > & vec,
            int L,
            int R )
```

A normal Quicksort implementation for vector arrays of doubles.

Normal Quicksort implementation for vector arrays.

Note

Sorted in Ascending Order.
Smallest value gets moved to index 0.
Largest value gets moved to the last index.

Parameters

vec	Vector array of doubles.
L	The starting index for the quicksort (inclusive).
R	The ending index for the quicksort (inclusive).

4.8.2.11 swap() [1/2]

```
void swap (
            vector< double > & fitnessList,
            vector< vector< double >> & matrix,
            int x,
            int y )
```

Swaps the fitness' and their corresponding vectors in the matrix.

Swap function for the Quicksort.

Parameters

fitnessList	The list of fitness values that correspond to each row of the matrix.
matrix	A matrix of double values.
x	The 1st index of the fitness/vector for the swap.
y	The 2nd index of the fitness/vector for the swap.

4.8.2.12 swap() [2/2]

```
void swap (
            vector< double > & v,
            int x,
            int y )
```

Swaps two values of a vector array of doubles.

Parameters

v	The vector in which values are swapped.
x	The 1st index of the fitness/vector for the swap.
y	The 2nd index of the fitness/vector for the swap.

4.9 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/Genetic↩ Algorithm.cpp File Reference

Implementation of the Genetic Algorithm.

```
#include "GeneticAlgorithm.h"
```

4.9.1 Detailed Description

Implementation of the Genetic Algorithm.

Author

Al Timofeyev

Date

April 25, 2019

4.10 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/Genetic↩ Algorithm.h File Reference

Implementation of the Genetic Algorithm.

```
#include <fstream>
#include <chrono>
#include "EA_Utilities.h"
```

Classes

- struct GA_Config

  Holds all the user defined variables. Genetic Algorithm Configuration Structure, where all user defined variables that are used to configure the Genetic Algorithm are stored.
- struct GA_Population

  Holds all the population information. Genetic Algorithm Population Structure, holds all the data related to the population of the Genetic Algorithm.
- struct GAAnalysis

  Genetic Algorithm Analysis Genetic Algorithm Analysis Structure, to keep track of the analysis performed on each population in the population list.
- class GeneticAlgorithm

4.10.1 Detailed Description

Implementation of the Genetic Algorithm.

Author

Al Timofeyev

Date

April 25, 2019

4.11 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/utilities.cpp File Reference

This utilities file is used as a helper file for ProcessFunctions.h and SearchAlgorithms.h, and to create matricies using the Mersenne Twister.

```
#include "utilities.h"
```

Functions

- vector< double > parseStringDbl (string str, string delimiter)

  Parses a string of numbers into a vector of doubles.
- vector< int > parseStringInt (string str, string delimiter)

  Parses a string of numbers into a vector of integers.
- vector< string > parseStringStr (string str, string delimiter)

  Parses a string of elements into a vector of strings.
- void prepForFunctionMatrix (vector< double > &setup)

  Resizes the vector to size 3.

4.11.1 Detailed Description

This utilities file is used as a helper file for ProcessFunctions.h and SearchAlgorithms.h, and to create matricies using the Mersenne Twister.

Author

Al Timofeyev

Date

April 15, 2019

4.11.2 Function Documentation

4.11.2.1 parseStringDbl()

```
vector<double> parseStringDbl (
            string str,
            string delimiter )
```

Parses a string of numbers into a vector of doubles.

Constructs and returns a vector of doubles, given a string list of numbers and a delimiter.

Note

The input string str MUST be a list of doubles!

Parameters

str	A string list of numbers.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with doubles that were extracted from the string list.

4.11.2.2 parseStringInt()

```
vector<int> parseStringInt (
            string str,
            string delimiter )
```

Parses a string of numbers into a vector of integers.

Constructs and returns a vector of integers, given a string list of numbers and a delimiter.

Note

The input string list MUST be a list of integers!

Parameters

str	A string list of numbers.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with integers that were extracted from the string list.

4.11.2.3 parseStringStr()

```
vector<string> parseStringStr (
            string str,
            string delimiter )
```

Parses a string of elements into a vector of strings.

Constructs and returns a vector of strings, given a string list of elements and a delimiter.

Parameters

str	A string list of characters.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with integers that were extracted from the string list.

4.11.2.4 prepForFunctionMatrix()

```
void prepForFunctionMatrix (
            vector< double > & setup )
```

Resizes the vector to size 3.

Resizes the given vector to size three in order to prep it for the matrix of a function. Because to generate a matrix, you only need 3 values: function ID, minimum bound, maximum bound.

Parameters

setup	The vector that's going to be resized for the matrix setup.

4.12 C:/Users/altim/Documents/School/CS471/Project3/EvolutionaryAlgorithms/utilities.h File Reference

This utilities file is used as a helper file for ProcessFunctions.h and SearchAlgorithms.h, and to create matricies using the Mersenne Twister.

```
#include <iostream>
#include <string>
#include <string.h>
#include <vector>
#include <cmath>
#include <random>
#include "BenchmarkFunctions.h"
```

Functions

- vector< double > parseStringDbl (string str, string delimiter)
  Parses a string of numbers into a vector of doubles.
- vector< int > parseStringInt (string str, string delimiter)
  Parses a string of numbers into a vector of integers.
- vector< string > parseStringStr (string str, string delimiter)
  Parses a string of elements into a vector of strings.
- void prepForFunctionMatrix (vector< double > &setup)
  Resizes the vector to size 3.

4.12.1 Detailed Description

This utilities file is used as a helper file for ProcessFunctions.h and SearchAlgorithms.h, and to create matricies using the Mersenne Twister.

Author

Al Timofeyev

Date

April 15, 2019

4.12.2 Function Documentation

4.12.2.1 parseStringDbl()

```
vector<double> parseStringDbl (
            string str,
            string delimiter )
```

Parses a string of numbers into a vector of doubles.

Parses a string of numbers into a vector of doubles.

Constructs and returns a vector of doubles, given a string list of numbers and a delimiter.

Note

The input string str MUST be a list of doubles!

Parameters

str	A string list of numbers.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with doubles that were extracted from the string list.

4.12.2.2 parseStringInt()

```
vector<int> parseStringInt (
            string str,
            string delimiter )
```

Parses a string of numbers into a vector of integers.

Parses a string of numbers into a vector of integers.

Constructs and returns a vector of integers, given a string list of numbers and a delimiter.

Note

The input string list MUST be a list of integers!

Parameters

str	A string list of numbers.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with integers that were extracted from the string list.

4.12.2.3 parseStringStr()

```
vector<string> parseStringStr (
            string str,
            string delimiter )
```

Parses a string of elements into a vector of strings.

Parses a string of characters into a vector of strings.

Constructs and returns a vector of strings, given a string list of elements and a delimiter.

Parameters

str	A string list of characters.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with integers that were extracted from the string list.

4.12.2.4 prepForFunctionMatrix()

```
void prepForFunctionMatrix (
            vector< double > & setup )
```

Resizes the vector to size 3.

Preps the setup vector for the matrix of a function by resizing to size 3.

Resizes the given vector to size three in order to prep it for the matrix of a function. Because to generate a matrix, you only need 3 values: function ID, minimum bound, maximum bound.

Parameters

setup	The vector that's going to be resized for the matrix setup.

Index