Genetic Algorithm's for Various Fitness Functions

1.0

Generated by Doxygen 1.8.14

Contents

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 files with brief descriptions:

Chapter 3

Class Documentation

3.1 Benchmark Class Reference

`#include <Benchmark.h>`

Public Types

- typedef double(Benchmark::* Fitness) (vector< double >)

Public Member Functions

- Benchmark ()
- Benchmark (int, int, Range)
- ~Benchmark ()
- void readText (const char *)
- void toCSV (const char *)
- void add_results (const vector< string > &)
- void clear_test_data ()
- void setSimulation (int count)
- void addHeader (string)
- vector< vector< double > > test_data ()
- void pseudo_random_matrix (int, int, Range)
- double schwefel (vector< double >)
- double first_de_jong (vector< double >)
- double rosenbrock (vector< double >)
- double rastrigin (vector< double >)
- double griewank (vector< double >)
- double sine_envelope_sine_wave (vector< double >)
- double stretch_v_sine_wave (vector< double >)
- double ackley_one (vector< double >)
- double ackley_two (vector< double >)
- double egg_holder (vector< double >)
- double rana (vector< double >)
- double pathological (vector< double >)
- double michalewicz (vector< double >)
- double masters_cosine_wave (vector< double >)
- double shekel_foxholes (vector< double >)

3.1.1 Detailed Description

Definition at line 13 of file Benchmark.h.

3.1.2 Member Typedef Documentation

3.1.2.1 Fitness

```
typedef double(Benchmark::* Benchmark::Fitness) (vector< double >)
```

Definition at line 23 of file Benchmark.h.

3.1.3 Constructor & Destructor Documentation

3.1.3.1 Benchmark() [1/2]

```
Benchmark::Benchmark ( )
```

This is the default constructor for Benchmark.

Definition at line 13 of file Benchmark.cpp.

3.1.3.2 Benchmark() [2/2]

```
Benchmark::Benchmark (
            int dimension,
            int max,
            Range rng )  [explicit]
```

This is an overloaded constructor for Benchmark that allows for creating simulation test data.

Parameters

dimension	amount of dimensions.
max	amount of simulations.
rng	this is the range (interval the values are in.

Definition at line 31 of file Benchmark.cpp.

3.1.3.3 ~Benchmark()

```
Benchmark::~Benchmark ( )
```

This is the destructor for Benchmark that will do clean up.

Definition at line 46 of file Benchmark.cpp.

3.1.4 Member Function Documentation

3.1.4.1 ackley_one()

```
double Benchmark::ackley_one (
            vector< double > x )
```

Find the benchmark result for Ackley One.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 340 of file Benchmark.cpp.

3.1.4.2 ackley_two()

```
double Benchmark::ackley_two (
            vector< double > x )
```

Find the benchmark result for Ackley Two.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 368 of file Benchmark.cpp.

3.1.4.3 add_results()

```
void Benchmark::add_results (
            const vector< string > & results )
```

This adds the computed results to Benchmark for converting to a CSV file.

Parameters

results	

Definition at line 84 of file Benchmark.cpp.

3.1.4.4 addHeader()

```
void Benchmark::addHeader (
            string hd )
```

This will add Time (sec) to the headers for the CSV file.

Definition at line 145 of file Benchmark.cpp.

3.1.4.5 clear_test_data()

```
void Benchmark::clear_test_data ( )
```

Clears the simulation test data.

Definition at line 93 of file Benchmark.cpp.

3.1.4.6 egg_holder()

```
double Benchmark::egg_holder (
            vector< double > x )
```

Find the benchmark result for Egg Holder.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 396 of file Benchmark.cpp.

3.1.4.7 first_de_jong()

```
double Benchmark::first_de_jong (
            vector< double > x )
```

Find the benchmark result for De Jong 1st.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 187 of file Benchmark.cpp.

3.1.4.8 griewank()

```
double Benchmark::griewank (
            vector< double > x )
```

Find the benchmark result for Griewank.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 260 of file Benchmark.cpp.

3.1.4.9 masters_cosine_wave()

```
double Benchmark::masters_cosine_wave (
            vector< double > x )
```

Find the benchmark result for Master's Cosine Wave.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 500 of file Benchmark.cpp.

3.1.4.10 michalewicz()

```
double Benchmark::michalewicz (
            vector< double > x )
```

Find the benchmark result for Michalewicz.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 475 of file Benchmark.cpp.

3.1.4.11 pathological()

```
double Benchmark::pathological (
            vector< double > x )
```

Find the benchmark result for Pathological.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 448 of file Benchmark.cpp.

3.1.4.12 pseudo_random_matrix()

```
void Benchmark::pseudo_random_matrix (
            int dimension,
            int max,
            Range rng )
```

This will randomly generate the simulation test data for the given parameters.

Parameters

dimension	amount of dimensions.
max	amount of simulations.
rng	this is the range (interval the values are in.

Definition at line 131 of file Benchmark.cpp.

3.1.4.13 rana()

```
double Benchmark::rana (
            vector< double > x )
```

Find the benchmark result for Rana.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 422 of file Benchmark.cpp.

3.1.4.14 rastrigin()

```
double Benchmark::rastrigin (
            vector< double > x )
```

Find the benchmark result for Rastrigin's Saddle.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 237 of file Benchmark.cpp.

3.1.4.15 readText()

```
void Benchmark::readText (
            const char * name )
```

This will read in a text file (∗.txt) for Shekel's Foxhole test.

Parameters

name	the name of the file

Definition at line 60 of file Benchmark.cpp.

3.1.4.16 rosenbrock()

```
double Benchmark::rosenbrock (
            vector< double > x )
```

Find the benchmark result for Rosenbrock.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 210 of file Benchmark.cpp.

3.1.4.17 schwefel()

```
double Benchmark::schwefel (
            vector< double > x )
```

Find the benchmark result for Schwefel.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 164 of file Benchmark.cpp.

3.1.4.18 setSimulation()

```
void Benchmark::setSimulation (
            int count )
```

This will set the amount of simulations being performed on the benchmark algorithm

Parameters

count	the amount of simulation performed

Definition at line 105 of file Benchmark.cpp.

3.1.4.19 shekel_foxholes()

```
double Benchmark::shekel_foxholes (
            vector< double > x )
```

Find the benchmark result for Shekel's Foxhole.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 528 of file Benchmark.cpp.

3.1.4.20 sine_envelope_sine_wave()

```
double Benchmark::sine_envelope_sine_wave (
            vector< double > x )
```

Find the benchmark result for Sine Envelope Sine Wave.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 286 of file Benchmark.cpp.

3.1.4.21 stretch_v_sine_wave()

```
double Benchmark::stretch_v_sine_wave (
            vector< double > x )
```

Find the benchmark result for Stretch V Sine Wave.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 313 of file Benchmark.cpp.

3.1.4.22 test_data()

```
vector< vector< double > > Benchmark::test_data ( )
```

This is a getter method to retrieve the simulation test data.

Returns

The the test data

Definition at line 117 of file Benchmark.cpp.

3.1.4.23 toCSV()

```
void Benchmark::toCSV (
            const char * name )
```

This will write all the test data to a comma-delimited text file (*.csv).

Parameters

name	the name of the file

Definition at line 72 of file Benchmark.cpp.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/Benchmark.h
- C:/Users/Shane Vance/CLionProjects/Optimization/Benchmark.cpp

3.2 ConstraintsFile Class Reference

```
#include <ConstraintsFile.h>
```

Public Member Functions

- ConstraintsFile ()
- ConstraintsFile (const char *)

Public Attributes

- vector< string > functionName
- vector< Range > range
- vector< Dimension > dimension
- Mutation mutation
- string extraFile
- int ns
- double cr
- double er
- int g_max
- double F
- int strategy

3.2.1 Detailed Description

Definition at line 48 of file ConstraintsFile.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ConstraintsFile() [1/2]

```
ConstraintsFile::ConstraintsFile ( )  [default]
```

This is the default constructor for the constraints class

3.2.2.2 ConstraintsFile() [2/2]

```
ConstraintsFile::ConstraintsFile (
            const char * name )  [explicit]
```

This will read constraints file with the values used for the Benchmark testing

Parameters

name	the name of the file you would like to read

Definition at line 20 of file ConstraintsFile.cpp.

3.2.3 Member Data Documentation

3.2.3.1 cr

```
double ConstraintsFile::cr
```

Definition at line 60 of file ConstraintsFile.h.

3.2.3.2 dimension

```
vector<Dimension> ConstraintsFile::dimension
```

Definition at line 56 of file ConstraintsFile.h.

3.2.3.3 er

```
double ConstraintsFile::er
```

Definition at line 61 of file ConstraintsFile.h.

3.2.3.4 extraFile

```
string ConstraintsFile::extraFile
```

Definition at line 58 of file ConstraintsFile.h.

3.2.3.5 F

```
double ConstraintsFile::F
```

Definition at line 63 of file ConstraintsFile.h.

3.2.3.6 functionName

```
vector<string> ConstraintsFile::functionName
```

Definition at line 54 of file ConstraintsFile.h.

3.2.3.7 g_max

```
int ConstraintsFile::g_max
```

Definition at line 62 of file ConstraintsFile.h.

3.2.3.8 mutation

```
Mutation ConstraintsFile::mutation
```

Definition at line 57 of file ConstraintsFile.h.

3.2.3.9 ns

```
int ConstraintsFile::ns
```

Definition at line 59 of file ConstraintsFile.h.

3.2.3.10 range

```
vector<Range> ConstraintsFile::range
```

Definition at line 55 of file ConstraintsFile.h.

3.2.3.11 strategy

```
int ConstraintsFile::strategy
```

Definition at line 64 of file ConstraintsFile.h.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.h
- C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.cpp

3.3 Dimension Struct Reference

```
#include <ConstraintsFile.h>
```

Public Member Functions

- Dimension ()=default
- Dimension (int lb, int ub)

Public Attributes

- int LB
- int UB

3.3.1 Detailed Description

Definition at line 27 of file ConstraintsFile.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Dimension() [1/2]

```
Dimension::Dimension ( )  [default]
```

3.3.2.2 Dimension() [2/2]

```
Dimension::Dimension (
            int lb,
            int ub )  [inline]
```

Definition at line 30 of file ConstraintsFile.h.

3.3.3 Member Data Documentation

3.3.3.1 LB

```
int Dimension::LB
```

Definition at line 31 of file ConstraintsFile.h.

3.3.3.2 UB

```
int Dimension::UB
```

Definition at line 31 of file ConstraintsFile.h.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.h

3.4 GeneticAlgorithms Class Reference

```
#include <GeneticAlgorithms.h>
```

Public Member Functions

- GeneticAlgorithms (const Benchmark &)
- vector< double > simpleGA (Benchmark::Fitness, int, int, Range, int, double, Mutation, double)
- vector< double > diffEvolution (Benchmark::Fitness, unsigned int, int, int, double, double, Range, int)

3.4.1 Detailed Description

Definition at line 55 of file GeneticAlgorithms.h.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 GeneticAlgorithms()

```
GeneticAlgorithms::GeneticAlgorithms (
            const Benchmark & newBM )  [explicit]
```

This sets up the genetic algorithm class.

Parameters

newBM	passes this for evaluating the functions

Definition at line 16 of file GeneticAlgorithms.cpp.

3.4.3 Member Function Documentation

3.4.3.1 diffEvolution()

```
vector< double > GeneticAlgorithms::diffEvolution (
            Benchmark::Fitness fn,
            unsigned int dim,
            int g_max,
            int np,
            double F,
            double cr,
            Range rng,
            int strategy )
```

This will evaluate the best solution of the fitness function using differential evolution and a selected strategy. The strategies vary from 1-10 and the first 5 are exponential and the other 5 are binomial. It goes through and selects random indexes to be evaluated with respect to its selected strategy.

Parameters

fn	the fitness function
dim	the amount of dimensions needed
g_max	the max amount of generations
np	the size of the population
F	the mutation rate
cr	the crossover rate
rng	the range of the fitness function
strategy	the selected strategy (1-10)

Returns

the best solution

Definition at line 93 of file GeneticAlgorithms.cpp.

3.4.3.2 simpleGA()

```
vector< double > GeneticAlgorithms::simpleGA (
            Benchmark::Fitness f,
            int ns,
            int dim,
            Range rng,
            int t_max,
            double cr,
            Mutation m,
            double er )
```

This is a version of the Genetic Algorithms (GA) that is known as Simple GA. This is the simplest algorithms of the GA's. This will find the best (most optimal solution) of the given fitness function with respect to its dimension and population.

Parameters

f	the fitness (cost) evaluator
ns	the size of the population
dim	the amount of chromosomes in the population (the dimension)
rng	the range of values
t_max	the max amount of generations
cr	the crossover rate
m	the mutation parameter
er	the elitism rate

Returns

the best solution

Definition at line 35 of file GeneticAlgorithms.cpp.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/GeneticAlgorithms.h
- C:/Users/Shane Vance/CLionProjects/Optimization/GeneticAlgorithms.cpp

3.5 LocalSearch Class Reference

`#include <LocalSearch.h>`

Public Member Functions

- LocalSearch ()
- LocalSearch (const Benchmark &)
- ~LocalSearch ()
- void setCount (int count)
- int getCount ()
- vector< double > randomWalk (Benchmark::Fitness, vector< double >, double *, int, Range)
- vector< double > localSearch (Benchmark::Fitness, const vector< double > &, double *, double, Range)
- vector< double > iterativeLocalSearch (Benchmark::Fitness, const vector< double > &, double *, double, int, Range)

3.5.1 Detailed Description

Definition at line 14 of file LocalSearch.h.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 LocalSearch() [1/2]

```
LocalSearch::LocalSearch ( )
```

This sets up the LocalSearch algorithm to be used.

Definition at line 13 of file LocalSearch.cpp.

3.5.2.2 LocalSearch() [2/2]

```
LocalSearch::LocalSearch (
            const Benchmark & bm )  [explicit]
```

This is a constructor that passes reference to the benchmark data

Parameters

bm	the Benchmark class that has the test data we need

Definition at line 22 of file LocalSearch.cpp.

3.5.2.3 ~LocalSearch()

```
LocalSearch::~LocalSearch ( )  [default]
```

This is the default destructor for LocalSearch

3.5.3 Member Function Documentation

3.5.3.1 getCount()

```
int LocalSearch::getCount ( )
```

This will get the amount of times it took to find the best solution

Returns

Definition at line 50 of file LocalSearch.cpp.

3.5.3.2 iterativeLocalSearch()

```
vector< double > LocalSearch::iterativeLocalSearch (
            Benchmark::Fitness f,
            const vector< double > & init,
            double * f_best,
            double delta,
            int t_max,
            Range rng )
```

This performs the iterative local search algorithm using the empirical gradient descent to find the best solution of the local optima

Parameters

f	the fitness function we want to use
init	the initial random vector
f_best	the initial best solution
delta	the delta we will use for the neighborhood search
t_max	the maximum amount of iterations we want to test with
rng	the range of values we will be testing with

Returns

Definition at line 156 of file LocalSearch.cpp.

3.5.3.3 localSearch()

```
vector< double > LocalSearch::localSearch (
            Benchmark::Fitness f,
            const vector< double > & init,
            double * f_best,
            double delta,
            Range rng )
```

This performs the local search algorithm using the empirical gradient descent to find the best solution of the local optima

Parameters

f	the fitness function we want to use
init	the initial random vector
f_best	the initial best solution
delta	the delta we will use for the neighborhood search
rng	the range of values we will be testing with

Returns

Definition at line 113 of file LocalSearch.cpp.

3.5.3.4 randomWalk()

```
vector< double > LocalSearch::randomWalk (
            Benchmark::Fitness f_cost,
            vector< double > arg,
            double * fitness_0,
            int itr,
            Range rng )
```

This performs the random walk (a.k.a blind worker) algorithm for finding the best solution for the local optima

Parameters

f_cost	the fitness function we want to use
arg	the vector we want to perform random walk on
fitness_0	the initial best fitness
itr	how many times we want to iterate
dim	the amount of dimensions
rng	the range of the values of lower-bound and upper-bound

Returns

Definition at line 69 of file LocalSearch.cpp.

3.5.3.5 setCount()

```
void LocalSearch::setCount (
            int count )
```

This will set the count for the current iteration

Parameters

count	initialize a value

Definition at line 40 of file LocalSearch.cpp.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/LocalSearch.h
- C:/Users/Shane Vance/CLionProjects/Optimization/LocalSearch.cpp

3.6 Mutation Struct Reference

```
#include <ConstraintsFile.h>
```

Public Attributes

- double rate
- double precision
- double range

3.6.1 Detailed Description

Definition at line 41 of file ConstraintsFile.h.

3.6.2 Member Data Documentation

3.6.2.1 precision

```
double Mutation::precision
```

Definition at line 44 of file ConstraintsFile.h.

3.6.2.2 range

```
double Mutation::range
```

Definition at line 45 of file ConstraintsFile.h.

3.6.2.3 rate

```
double Mutation::rate
```

Definition at line 43 of file ConstraintsFile.h.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.h

3.7 Parent Struct Reference

```
#include <GeneticAlgorithms.h>
```

Public Attributes

- vector< double > one
- vector< double > two

3.7.1 Detailed Description

Definition at line 47 of file GeneticAlgorithms.h.

3.7.2 Member Data Documentation

3.7.2.1 one

```
vector<double> Parent::one
```

Definition at line 50 of file GeneticAlgorithms.h.

3.7.2.2 two

```
vector<double> Parent::two
```

Definition at line 51 of file GeneticAlgorithms.h.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/GeneticAlgorithms.h

3.8 Population Struct Reference

```
#include <GeneticAlgorithms.h>
```

Public Member Functions

- Population (vector< double > genes)
- bool operator< (const Population &population) const

Public Attributes

- vector< double > genome
- double cost {}

3.8.1 Detailed Description

Definition at line 19 of file GeneticAlgorithms.h.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Population()

```
Population::Population (
            vector< double > genes )  [inline], [explicit]
```

This setup up the population structure

Parameters

genes	

Definition at line 28 of file GeneticAlgorithms.h.

3.8.3 Member Function Documentation

3.8.3.1 operator<()

```
bool Population::operator< (
            const Population & population ) const  [inline]
```

Definition at line 33 of file GeneticAlgorithms.h.

3.8.4 Member Data Documentation

3.8.4.1 cost

```
double Population::cost {}
```

Definition at line 30 of file GeneticAlgorithms.h.

3.8.4.2 genome

```
vector<double> Population::genome
```

Definition at line 29 of file GeneticAlgorithms.h.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/GeneticAlgorithms.h

3.9 Range Struct Reference

```
#include <ConstraintsFile.h>
```

Public Member Functions

- Range ()=default
- Range (double lb, double ub)

Public Attributes

- double LB
- double UB

3.9.1 Detailed Description

Definition at line 16 of file ConstraintsFile.h.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 Range() [1/2]

```
Range::Range ( )  [default]
```

3.9.2.2 Range() [2/2]

```
Range::Range (
            double lb,
            double ub )  [inline]
```

Definition at line 19 of file ConstraintsFile.h.

3.9.3 Member Data Documentation

3.9.3.1 LB

```
double Range::LB
```

Definition at line 20 of file ConstraintsFile.h.

3.9.3.2 UB

```
double Range::UB
```

Definition at line 20 of file ConstraintsFile.h.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.h

Chapter 4

File Documentation

4.1 C:/Users/Shane Vance/CLionProjects/Optimization/Benchmark.cpp File Reference

```
#include "Benchmark.h"
```

4.2 C:/Users/Shane Vance/CLionProjects/Optimization/Benchmark.h File Reference

```
#include "ConstraintsFile.h"
#include "stdafx.h"
```

Classes

- class Benchmark

4.3 C:/Users/Shane Vance/CLionProjects/Optimization/BenchmarkClient.cpp File Reference

```
#include "BenchmarkClient.h"
```

Functions

- void run ()
- string toString (double val)
- void mainMenu (int strategy)
- void compute (Benchmark::Fitness f, Benchmark ∗bm, GA ga, CF cf, int selection, unsigned int i)

4.3.1 Function Documentation

4.3.1.1 compute()

```
void compute (
            Benchmark::Fitness f,
            Benchmark * bm,
            GA ga,
            CF cf,
            int selection,
            unsigned int i )
```

This will compute each of the function for n-dimensions and m-simulation

Parameters

f	the fitness function we want to use
bm	the Benchmark class that contains the test functions
ga	the GeneticAlgorithm (GA) class that contains the GA functions
cf	this is the constraints file
selection	this is the option that is selected by the user for the program to run
i	this is the index for which is used by the constraints file

Definition at line 137 of file BenchmarkClient.cpp.

4.3.1.2 mainMenu()

```
void mainMenu (
            int strategy )
```

This will give a prompt for the user giving them options to select from to execute a program

Definition at line 113 of file BenchmarkClient.cpp.

4.3.1.3 run()

```
void run ( )
```

This will be used by main.cpp to run the program. This will perform various evaluation at the command of the user.

Definition at line 13 of file BenchmarkClient.cpp.

4.3.1.4 toString()

```
string toString (
            double val )
```

This converts a double to a string in scientific notation

Parameters

val	pass a double that you would like to convert to a double

Returns

Definition at line 102 of file BenchmarkClient.cpp.

4.4 C:/Users/Shane Vance/CLionProjects/Optimization/BenchmarkClient.h File Reference

```
#include "stdafx.h"
#include "ConstraintsFile.h"
#include "Benchmark.h"
#include "GeneticAlgorithms.h"
```

Typedefs

- typedef ConstraintsFile CF
- typedef GeneticAlgorithms GA

Functions

- void run ()
- void mainMenu (int)
- string toString (double)
- void compute (Benchmark::Fitness, Benchmark ∗, GA, CF, int, unsigned int)

4.4.1 Typedef Documentation

4.4.1.1 CF

```
typedef ConstraintsFile CF
```

Definition at line 21 of file BenchmarkClient.h.

4.4.1.2 GA

```
typedef GeneticAlgorithms GA
```

Definition at line 29 of file BenchmarkClient.h.

4.4.2 Function Documentation

4.4.2.1 compute()

```
void compute (
            Benchmark::Fitness f,
            Benchmark * bm,
            GA ga,
            CF cf,
            int selection,
            unsigned int i )
```

This will compute each of the function for n-dimensions and m-simulation

Parameters

f	the fitness function we want to use
bm	the Benchmark class that contains the test functions
ga	the GeneticAlgorithm (GA) class that contains the GA functions
cf	this is the constraints file
selection	this is the option that is selected by the user for the program to run
i	this is the index for which is used by the constraints file

Definition at line 137 of file BenchmarkClient.cpp.

4.4.2.2 mainMenu()

```
void mainMenu (
            int strategy )
```

This will give a prompt for the user giving them options to select from to execute a program

Definition at line 113 of file BenchmarkClient.cpp.

4.4.2.3 run()

```
void run ( )
```

This will be used by main.cpp to run the program. This will perform various evaluation at the command of the user.

Definition at line 13 of file BenchmarkClient.cpp.

4.4.2.4 toString()

```
string toString (
            double val )
```

This converts a double to a string in scientific notation

Parameters

val	pass a double that you would like to convert to a double

Returns

Definition at line 102 of file BenchmarkClient.cpp.

4.5 C:/Users/Shane Vance/CLionProjects/Optimization/cmake-build-debug/CMake↩ Files/3.10.3/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define C_DIALECT

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char const ∗ info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const ∗ info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const ∗ info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char ∗ info_language_dialect_default

4.5.1 Macro Definition Documentation

4.5.1.1 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 468 of file CMakeCCompilerId.c.

4.5.1.2 C_DIALECT

```
#define C_DIALECT
```

Definition at line 552 of file CMakeCCompilerId.c.

4.5.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 288 of file CMakeCCompilerId.c.

4.5.1.4 DEC

```
#define DEC(
              n )
```

Value:

```
('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 472 of file CMakeCCompilerId.c.

4.5.1.5 HEX

```
#define HEX(
              n )
```

Value:

```
('0' + ((n)>>28 & 0xF)), \
  ('0' + ((n)>>24 & 0xF)), \
  ('0' + ((n)>>20 & 0xF)), \
  ('0' + ((n)>>16 & 0xF)), \
  ('0' + ((n)>>12 & 0xF)), \
  ('0' + ((n)>>8  & 0xF)), \
  ('0' + ((n)>>4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 483 of file CMakeCCompilerId.c.

4.5.1.6 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 405 of file CMakeCCompilerId.c.

4.5.1.7 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

Definition at line 309 of file CMakeCCompilerId.c.

4.5.1.8 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

Definition at line 308 of file CMakeCCompilerId.c.

4.5.2 Function Documentation

4.5.2.1 main()

```
int main (
             int argc,
             char * argv[] )
```

Definition at line 572 of file CMakeCCompilerId.c.

4.5.3 Variable Documentation

4.5.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 543 of file CMakeCCompilerId.c.

4.5.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 295 of file CMakeCCompilerId.c.

4.5.3.3 info_language_dialect_default

```
const char* info_language_dialect_default
```

Initial value:

```
=
  "INFO" ":" "dialect_default[" C_DIALECT "]"
```

Definition at line 561 of file CMakeCCompilerId.c.

4.5.3.4 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 542 of file CMakeCCompilerId.c.

4.6 C:/Users/Shane Vance/CLionProjects/Optimization/cmake-build-debug/CMakeFiles/3.10.3/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define COMPILER_ID ""
- #define STRINGIFY_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY_HELPER(X)
- #define PLATFORM_ID
- #define ARCHITECTURE_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX_STD __cplusplus

Functions

- int main (int argc, char *argv[])

Variables

- char const * info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const * info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const * info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
- const char * info_language_dialect_default

4.6.1 Macro Definition Documentation

4.6.1.1 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 453 of file CMakeCXXCompilerId.cpp.

4.6.1.2 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 273 of file CMakeCXXCompilerId.cpp.

4.6.1.3 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 536 of file CMakeCXXCompilerId.cpp.

4.6.1.4 DEC

```
#define DEC(
              n )
```

Value:

```
('0' + (((n) / 10000000)%10)), \
  ('0' + (((n) / 1000000)%10)),  \
  ('0' + (((n) / 100000)%10)),   \
  ('0' + (((n) / 10000)%10)),    \
  ('0' + (((n) / 1000)%10)),     \
  ('0' + (((n) / 100)%10)),      \
  ('0' + (((n) / 10)%10)),       \
  ('0' +  ((n) % 10))
```

Definition at line 457 of file CMakeCXXCompilerId.cpp.

4.6.1.5 HEX

```
#define HEX(
              n )
```

Value:

```
('0' + ((n)>>28 & 0xF)), \
  ('0' + ((n)>>24 & 0xF)), \
  ('0' + ((n)>>20 & 0xF)), \
  ('0' + ((n)>>16 & 0xF)), \
  ('0' + ((n)>>12 & 0xF)), \
  ('0' + ((n)>>8  & 0xF)), \
  ('0' + ((n)>>4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

Definition at line 468 of file CMakeCXXCompilerId.cpp.

4.6.1.6 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 390 of file CMakeCXXCompilerId.cpp.

4.6.1.7 STRINGIFY

```
#define STRINGIFY(
              X ) STRINGIFY_HELPER(X)
```

Definition at line 294 of file CMakeCXXCompilerId.cpp.

4.6.1.8 STRINGIFY_HELPER

```
#define STRINGIFY_HELPER(
              X ) #X
```

Definition at line 293 of file CMakeCXXCompilerId.cpp.

4.6.2 Function Documentation

4.6.2.1 main()

```
int main (
             int argc,
             char * argv[] )
```

Definition at line 553 of file CMakeCXXCompilerId.cpp.

4.6.3 Variable Documentation

4.6.3.1 info_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 528 of file CMakeCXXCompilerId.cpp.

4.6.3.2 info_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 280 of file CMakeCXXCompilerId.cpp.

4.6.3.3 info_language_dialect_default

```
const char* info_language_dialect_default
```

Initial value:

```
= "INFO" ":" "dialect_default["



  "98"
"]"
```

Definition at line 539 of file CMakeCXXCompilerId.cpp.

4.6.3.4 info_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 527 of file CMakeCXXCompilerId.cpp.

4.7 C:/Users/Shane Vance/CLionProjects/Optimization/cmake-build-debug/CMakeFiles/feature_tests.c File Reference

Functions

- int main (int argc, char **argv)

Variables

- const char features [ ]

4.7.1 Function Documentation

4.7.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

Definition at line 34 of file feature_tests.c.

4.7.2 Variable Documentation

4.7.2.1 features

```
const char features[]
```

Definition at line 2 of file feature_tests.c.

4.8 C:/Users/Shane Vance/CLionProjects/Optimization/cmake-build-debug/CMakeFiles/feature_tests.cxx File Reference

Functions

- int main (int argc, char **argv)

Variables

- const char features [ ]

4.8.1 Function Documentation

4.8.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

Definition at line 405 of file feature_tests.cxx.

4.8.2 Variable Documentation

4.8.2.1 features

```
const char features[]
```

Definition at line 2 of file feature_tests.cxx.

4.9 C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.cpp File Reference

```
#include "ConstraintsFile.h"
```

4.10 C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.h File Reference

```
#include "stdafx.h"
```

Classes

- struct Range
- struct Dimension
- struct Mutation
- class ConstraintsFile

Typedefs

- typedef struct Range Range
- typedef struct Dimension Dimension
- typedef struct Mutation Mutation

4.10.1 Typedef Documentation

4.10.1.1 Dimension

```
Dimension
```

4.10.1.2 Mutation

```
Mutation
```

This will create the mutation structure containing all the parameter necessary for mutation in the GA algorithms.

4.10.1.3 Range

```
typedef struct Range Range
```

4.11 C:/Users/Shane Vance/CLionProjects/Optimization/GeneticAlgorithms.cpp File Reference

```
#include "GeneticAlgorithms.h"
```

4.12 C:/Users/Shane Vance/CLionProjects/Optimization/GeneticAlgorithms.h File Reference

```
#include "Benchmark.h"
```

Classes

- struct Population
- struct Parent
- class GeneticAlgorithms

Typedefs

- typedef struct Population NewPopulation
- typedef struct Parent Child

4.12.1 Typedef Documentation

4.12.1.1 Child

`Parent` and `Child`

This creates an individual whether it be a child or parent.

4.12.1.2 NewPopulation

`Population` and `NewPopulation`

This defines the new population

4.13 C:/Users/Shane Vance/CLionProjects/Optimization/LocalSearch.cpp File Reference

```
#include "LocalSearch.h"
```

4.14 C:/Users/Shane Vance/CLionProjects/Optimization/LocalSearch.h File Reference

```
#include "ConstraintsFile.h"
#include "Benchmark.h"
#include "stdafx.h"
```

Classes

- class LocalSearch

4.15 C:/Users/Shane Vance/CLionProjects/Optimization/main.cpp File Reference

```
#include "stdafx.h"
#include "BenchmarkClient.h"
```

Macros

- #define __STRICT_ANSI__

Functions

- int main (int argc, char **argv)

4.15.1 Macro Definition Documentation

4.15.1.1 __STRICT_ANSI__

```
#define __STRICT_ANSI__
```

Definition at line 8 of file main.cpp.

4.15.2 Function Documentation

4.15.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

Definition at line 12 of file main.cpp.

4.16 C:/Users/Shane Vance/CLionProjects/Optimization/stdafx.cpp File Reference

```
#include "stdafx.h"
```

4.17 C:/Users/Shane Vance/CLionProjects/Optimization/stdafx.h File Reference

```
#include <iostream>
#include <sstream>
#include <cmath>
#include <ctime>
#include <cstring>
#include <vector>
#include <string>
#include <cstdio>
#include <random>
#include <algorithm>
#include <thread>
#include <future>
#include <limits>
#include <unistd.h>
```