CS471 - Project 2 - Search Algorithms

2.0

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 FunctionAnalysis Struct Reference

Function Analysis Function Analysis Structure, to keep track of the analysis performed on each FunctionData structure. Basically, it compiles and holds the averages of the calculations performed for each function.

```
#include <DataStructs.h>
```

Public Attributes

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

3.1.1 Detailed Description

Function Analysis Function Analysis Structure, to keep track of the analysis performed on each FunctionData structure. Basically, it compiles and holds the averages of the calculations performed for each function.

3.1.2 Member Data Documentation

3.1.2.1 avgFunctionFitness

```
vector<double> FunctionAnalysis::avgFunctionFitness
```

List of the average fitness per FunctionData structure.

3.1.2.2 functionIDs

```
vector<int> FunctionAnalysis::functionIDs
```

List of function IDs.

3.1.2.3 header

```
string FunctionAnalysis::header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median
```

Header used when saving the data.

3.1.2.4 medianFunctionFitness

```
vector<double> FunctionAnalysis::medianFunctionFitness
```

List of the Median fitness from each FunctionData structure.

3.1.2.5 processTimes

```
vector<double> FunctionAnalysis::processTimes
```

List of process times in ms for all functions.

3.1.2.6 ranges

```
vector<vector<double> > FunctionAnalysis::ranges
```

List of ranges for each fitness result in resultsOfFunctions.

3.1.2.7 standardDeviation

```
vector<double> FunctionAnalysis::standardDeviation
```

List of standard fitness deviations.

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

- C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/DataStructs.h

3.2 FunctionData Struct Reference

Function Data Function Data Structure, to keep track of all the data used for the Benchmark Functions.

```
#include <DataStructs.h>
```

Public Attributes

- int functionID
- double **minBound**
- double maxBound
- vector< double > fitness
- vector< vector< double > > functionMatrix
- double timeToExecute = -1.0

3.2.1 Detailed Description

Function Data Function Data Structure, to keep track of all the data used for the Benchmark Functions.

3.2.2 Member Data Documentation

3.2.2.1 fitness

```
vector<double> FunctionData::fitness
```

The list of fitness for each vector in the matrix.

3.2.2.2 functionID

```
int FunctionData::functionID
```

The ID used to determine which of the 18 Benchmark Functions to use.

3.2.2.3 functionMatrix

```
vector<vector<double> > FunctionData::functionMatrix
```

The matrix of double vectors.

3.2.2.4 maxBound

```
double FunctionData::maxBound
```

The max and min bound used for the matrix.

3.2.2.5 timeToExecute

```
double FunctionData::timeToExecute = -1.0
```

This is time in ms to process all 30 rows.

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

- C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/DataStructs.h

3.3 ProcessFunctions Class Reference

A class used to process matrices against Benchmark Functions and analyze the results.

```
#include "ProcessFunctions.h"
```

Public Member Functions

- ProcessFunctions ()

  The default constructor for the ProcessFunctions class. The default constructor only initializes the numOfDimensions variable to 0;.
- void setNumOfDimensions (int dimensions)

  Sets the number of dimensions for the ProcessFunctions object.
- int getNumOfDimensions ()

  Returns the number of dimensions used for the matrix.
- void constructMatrix ()

  Generates a matrix using Mersenne Twister.
- void constructMatrix (int funcID, double minBoundary, double maxBoundary)

  Generates a matrix using Mersenne Twister.
- void calculateFitnessOfAllMatrices ()

  Calculates the fitness of all Matrices in resultsOfFunctions vector.
- void analyzeAllFunctionResults ()

  Analyzes all the results from each FunctionData structure in resultsOfFunctions.
- void performAllSearchAlgorithms ()

  Performs all search algorithms for each FunctionData structure in resultsOfFunctions.
- void analyzeAllSearchAlgorithmResults ()

  Analyzes all the results from the search algorithms.
- void saveAllMatricesToFile (string configFilename)

  Saves all the matrices in resultsOfFunctions vector to files.
- void saveAllProcessedFunctionDataToFile (string configFilename)

  Saves all the data in resultsOfFunctions to files.
- void saveAllAnalyzedDataToFile (string configFilename)

  Saves all analyzed data in analysis to file.
- void saveAllAnalyzedDataToSpecificFile (string filename)

  Saves all analyzed data in analysis to user-specified file.
- void saveAllAnalyzedSearchAlgDataToFile (string configFilename)

  Saves all analyzed data in searchAlgAnalysis to file.
- void printAllFunctionIDs ()

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

  Prints all the FunctionData structures in resultsOfFunctions.
- void printFunctionResultsAnalysis ()

  Prints all the Analysis Results in analysis.

Private Member Functions

- FunctionData generateMatrix (double minBoundary, double maxBoundary)

  Generates a DEFAULT_NUM_OF_VECTORS by numOfDimensions matrix using Mersenne Twister.
- void calculateMatrixFitness (FunctionData &data)

  Calculates the fitness of all vectors of a matrix.
- void analyzeFunctionResults (FunctionData &data)
- double getMinFitness (FunctionData &data)

  Returns the minimum fitness of the data in FunctionData struct.
- double getMaxFitness (FunctionData &data)

  Returns the maximum fitness of the data in FunctionData struct.
- void saveFunctionMatrixToFile (string filename, FunctionData &data)

  Saves the matrix of the FunctionData to file.
- void saveAllFunctionDataToFile (string filename, FunctionData &data)

  Saves all the data of the function to file.
- void quicksortFuncData (FunctionData &data, int L, int R)

  Quicksort Algorithm for sorting the fitness and matrix using the fitness.
- void swapFuncData (FunctionData &data, int x, int y)

  Specialized swap function for the quicksort algorithm.

Private Attributes

- int **numOfDimensions**
- vector< FunctionData > **resultsOfFunctions**
- FunctionAnalysis **analysis**
- vector< SearchAlgorithmResults > **searchAlgResults**
- SearchAlgorithmResultsAnalysis **searchAlgAnalysis**

3.3.1 Detailed Description

A class used to process matrices against Benchmark Functions and analyze the results.

Author

Al Timofeyev

Date

April 4, 2019

3.3.2 Member Function Documentation

3.3.2.1 analyzeAllFunctionResults()

```
void ProcessFunctions::analyzeAllFunctionResults ( )
```

Analyzes all the results from each FunctionData structure in resultsOfFunctions.

Analyzes all the results from resultsOfFunctions.

3.3.2.2 analyzeAllSearchAlgorithmResults()

```
void ProcessFunctions::analyzeAllSearchAlgorithmResults ( )
```

Analyzes all the results from the search algorithms.

Analyzes all the results from the search algorithms.

3.3.2.3 analyzeFunctionResults()

```
void ProcessFunctions::analyzeFunctionResults (
            FunctionData & data )  [private]
```

Analyzes the results of the functions.

Parameters

data	Analyzes the results of the functions.

3.3.2.4 calculateFitnessOfAllMatrices()

```
void ProcessFunctions::calculateFitnessOfAllMatrices ( )
```

Calculates the fitness of all Matrices in resultsOfFunctions vector.

Calculates Fitness for all matrices in resultsOfFunctions.

3.3.2.5 calculateMatrixFitness()

```
void ProcessFunctions::calculateMatrixFitness (
            FunctionData & data )  [private]
```

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 in a FunctionData structure. All the fitness results are stored in the fitness vector variable of the same FunctionData structure.

Note

> This function makes a call to utilities.h --> calculateFitnessOfMatrix().

Parameters

data	The FunctionData structure that contains the matrix.

3.3.2.6 constructMatrix() [1/2]

```
void ProcessFunctions::constructMatrix ( )
```

Generates a matrix using Mersenne Twister.

Uses all default constants, or previously user-set dimensions.

A matrix is constructed using the default number of dimensions, or a previously user-set number of dimensions, and the default minimum and maximum bound. Saves the constructed matrix to variable resultsOfFunctions.

3.3.2.7 constructMatrix() [2/2]

```
void ProcessFunctions::constructMatrix (
            int funcID,
            double minBoundary,
            double maxBoundary )
```

Generates a matrix using Mersenne Twister.

Uses default number of dimensions.

A matrix is constructed using the default value of 30 dimensions, or a previously user-set number of dimensions, and a user-provided minimum and maximum bound. Saves the constructed matrix to variable resultsOfFunctions.

Parameters

funcID	The function ID for which Benchmark Function the matrix is generated for.
minBoundary,maxBoundary	The minimum and maximum boundaries for the values in the matrix.

3.3.2.8 generateMatrix()

```
FunctionData ProcessFunctions::generateMatrix (
            double minBoundary,
            double maxBoundary )  [private]
```

Generates a DEFAULT_NUM_OF_VECTORS by numOfDimensions matrix using Mersenne Twister.

Generates a matrix using min/max boundaries.

A matrix is constructed using the specified number of dimensions stored in numOfDimensions and a user-provided minimum and maximum bound.

Note

> DEFAULT_NUM_OF_VECTORS is currently set to 30 (as of April 4, 2019).
> This function makes a call to utilities.h --> createMatrix().

Parameters

minBoundary,maxBoundary	The max/min boundaries are the range in which to generate numbers.

Returns

The struct that contains the constructed matrix and an empty list of function fitness results.

3.3.2.9 getMaxFitness()

```
double ProcessFunctions::getMaxFitness (
            FunctionData & data )  [private]
```

Returns the maximum fitness of the data in FunctionData struct.

Returns the maximum fitness of data.

Parameters

data	The FunctionData structure that contains a list of fitness values.

Returns

The Maximum fitness in FunctionaData data structure.

3.3.2.10 getMinFitness()

```
double ProcessFunctions::getMinFitness (
            FunctionData & data )  [private]
```

Returns the minimum fitness of the data in FunctionData struct.

Returns the minimum fitness of data.

Parameters

data	The FunctionData structure that contains a list of fitness values.

Returns

The Minimum fitness in FunctionaData data structure.

3.3.2.11 getNumOfDimensions()

```
int ProcessFunctions::getNumOfDimensions ( )
```

Returns the number of dimensions used for the matrix.

Returns the number of dimensions.

Returns

The value stored in the numOfDimensions variable.

3.3.2.12 performAllSearchAlgorithms()

```
void ProcessFunctions::performAllSearchAlgorithms ( )
```

Performs all search algorithms for each FunctionData structure in resultsOfFunctions.

Executes all the search algorithms.

Note

This function makes a call to utilities.h --> quicksort().

3.3.2.13 printAllFunctionIDs()

```
void ProcessFunctions::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 funtions they reference, to the screen.

3.3.2.14 printFunctionResults()

```
void ProcessFunctions::printFunctionResults ( )
```

Prints all the FunctionData structures in resultsOfFunctions.

Prints all the FunctionData structures in resultsOfFunctions.

3.3.2.15 printFunctionResultsAnalysis()

```
void ProcessFunctions::printFunctionResultsAnalysis ( )
```

Prints all the Analysis Results in analysis.

Prints all the Analysis Results in analysis.

3.3.2.16 quicksortFuncData()

```
void ProcessFunctions::quicksortFuncData (
            FunctionData & data,
            int L,
            int R ) [private]
```

Quicksort Algorithm for sorting the fitness and matrix using the fitness.

Special quicksort implementation.

Note

Smallest (minimum) fitness gets moved to index 0, along with its vector.
Largest (maximum) fitness gets moved to the last index, along with its vector.

Parameters

data	The FunctionData struct that hold the fitness vector and matrix.
L	The starting index for the quicksort (inclusive).
R	The ending index for the quicksort (inclusive).

3.3.2.17 saveAllAnalyzedDataToFile()

```
void ProcessFunctions::saveAllAnalyzedDataToFile (
            string configFilename )
```

Saves all analyzed data in analysis to file.

Saves all analyzed data in analysis to file.

Note

This function makes a call to utilities.h --> parseStringStr().

Parameters

configFilename	The configuration file from which data was generated.

3.3.2.18 saveAllAnalyzedDataToSpecificFile()

```
void ProcessFunctions::saveAllAnalyzedDataToSpecificFile (
             string filename )
```

Saves all analyzed data in analysis to user-specified file.

Saves all analyzed data in analysis to user-specified file.

Parameters

filename	The name of the file where to save the analysis, preferably an Excel (.csv) file.

3.3.2.19 saveAllAnalyzedSearchAlgDataToFile()

```
void ProcessFunctions::saveAllAnalyzedSearchAlgDataToFile (
             string configFilename )
```

Saves all analyzed data in searchAlgAnalysis to file.

Saves all analyzed data in searchAlgAnalysis to file.

Note

This function makes a call to utilities.h --> parseStringStr().

Parameters

configFilename	Configuration file from which data was generated.

3.3.2.20 saveAllFunctionDataToFile()

```
void ProcessFunctions::saveAllFunctionDataToFile (
             string filename,
             FunctionData & data )  [private]
```

Saves all the data of the function to file.

Saves the results of the function and it's data to file.

Parameters

filename	The filename where to store the matrix. Should be a Excel file (.csv).
data	A FunctionData struct that contains all the data of the function, including the matrix that was used as well as the fitness result of that function.

3.3.2.21 saveAllMatricesToFile()

```
void ProcessFunctions::saveAllMatricesToFile (
            string configFilename )
```

Saves all the matrices in resultsOfFunctions vector to files.

Saves all the matrices in resultsOfFunctions to files.

Note

This function makes a call to utilities.h --> parseStringStr().

Parameters

configFilename	The configuration file from which data was generated.

3.3.2.22 saveAllProcessedFunctionDataToFile()

```
void ProcessFunctions::saveAllProcessedFunctionDataToFile (
            string configFilename )
```

Saves all the data in resultsOfFunctions to files.

Saves all the data in resultsOfFunctions to files.

Note

This function makes a call to utilities.h --> parseStringStr().

Parameters

configFilename	The configuration file from which data was generated.

3.3.2.23 saveFunctionMatrixToFile()

```
void ProcessFunctions::saveFunctionMatrixToFile (
            string filename,
            FunctionData & data )  [private]
```

Saves the matrix of the FunctionData to file.

Saves the matrix to file.

Parameters

filename	The filename where to store the matrix. Should be a Excel file (.csv).
data	A FunctionData struct that contains all the data of the function, including the matrix that was used as well as the fitness result of that function.

3.3.2.24 setNumOfDimensions()

```
void ProcessFunctions::setNumOfDimensions (
            int dimensions )
```

Sets the number of dimensions for the ProcessFunctions object.

Sets the number of dimensions.

After setting the new number of dimensions, the resultsOfFunctions vector that held all the previous data, for the previous number of dimensions, is also reset to 0, and a new FunctionAnalysis struct is assigned to analysis.

Parameters

dimensions	The number of dimensions in the matrix data (dimensions = size of each vector in the matrix).

3.3.2.25 swapFuncData()

```
void ProcessFunctions::swapFuncData (
            FunctionData & data,
            int x,
            int y )  [private]
```

Specialized swap function for the quicksort algorithm.

Parameters

data	The FunctionData struct that hold the fitness vector and matrix.
x	The 1st index of the fitness/vector for the swap.
y	The 2nd index of the fitness/vector for the swap.

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

- C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/ProcessFunctions.h
- C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/ProcessFunctions.cpp

3.4 SearchAlgorithmResults Struct Reference

Results of Search Algorithms. Search Algorithm Results Structure, to keep track of the search algorithm output.

```
#include <DataStructs.h>
```

Public Attributes

- int functionID
- double fitnessBS
- double timeBS
- double fitnessLS
- double timeLS
- vector< double > fitnessILS
- double timeILS

3.4.1 Detailed Description

Results of Search Algorithms. Search Algorithm Results Structure, to keep track of the search algorithm output.

3.4.2 Member Data Documentation

3.4.2.1 fitnessBS

```
double SearchAlgorithmResults::fitnessBS
```

The best fitness found in Blinnd Search Algorithm.

3.4.2.2 fitnessILS

```
vector<double> SearchAlgorithmResults::fitnessILS
```

List of the best fitness' found in Iterative Local Search Algorithm.

3.4.2.3 fitnessLS

```
double SearchAlgorithmResults::fitnessLS
```

The best fitness found in Local Search Algorithm.

3.4.2.4 functionID

```
int SearchAlgorithmResults::functionID
```

The ID used to determine which of the 18 Benchmark Functions to use.

3.4.2.5 timeBS

```
double SearchAlgorithmResults::timeBS
```

The time it took to execute the Blind Search Algorithm in milliseconds.

3.4.2.6 timeILS

```
double SearchAlgorithmResults::timeILS
```

The time it took to execute the Iterative Local Search Algorithm in milliseconds.

3.4.2.7 timeLS

```
double SearchAlgorithmResults::timeLS
```

The time it took to execute the Local Search Algorithm in milliseconds.

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

- C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/DataStructs.h

3.5 SearchAlgorithmResultsAnalysis Struct Reference

Search Algorithm Analysis Search Algorithm Analysis Structure, to keep track of the analysis performed on each SearchAlgorithmResults structure. Basically, it compiles and holds the averages of the calculations performed for each search algorithm.

```
#include <DataStructs.h>
```

Public Attributes

- string **mainHeader** = " ,Blind Search, , , , , ,Local Search, , , , , ,Iterative Local Search, , , , ,\n"
- string **header** = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time(ms),Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time(ms),Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time(ms)\n"
- vector< int > functionIDs
- vector< double > avgBSFitness
- vector< double > standardDeviationBS
- vector< vector< double > > rangesBS
- vector< double > medianBSFitness
- vector< double > processTimesBS
- vector< double > avgLSFitness
- vector< double > standardDeviationLS
- vector< vector< double > > rangesLS
- vector< double > medianLSFitness
- vector< double > processTimesLS
- vector< double > avgILSFitness
- vector< double > standardDeviationILS
- vector< vector< double > > rangesILS
- vector< double > medianILSFitness
- vector< double > processTimesILS

3.5.1 Detailed Description

Search Algorithm Analysis Search Algorithm Analysis Structure, to keep track of the analysis performed on each SearchAlgorithmResults structure. Basically, it compiles and holds the averages of the calculations performed for each search algorithm.

3.5.2 Member Data Documentation

3.5.2.1 avgBSFitness

```
vector<double> SearchAlgorithmResultsAnalysis::avgBSFitness
```

List of the average Blind Search fitness per SearchAlgorithmResults structure.

3.5.2.2 avgILSFitness

```
vector<double> SearchAlgorithmResultsAnalysis::avgILSFitness
```

List of the average Iterative Local Search fitness per SearchAlgorithmResults structure.

3.5.2.3 avgLSFitness

```
vector<double> SearchAlgorithmResultsAnalysis::avgLSFitness
```

List of the average Local Search fitness per SearchAlgorithmResults structure.

3.5.2.4 functionIDs

```
vector<int> SearchAlgorithmResultsAnalysis::functionIDs
```

List of function IDs.

3.5.2.5 medianBSFitness

```
vector<double> SearchAlgorithmResultsAnalysis::medianBSFitness
```

List of the Median Blind Search fitness from each SearchAlgorithmResults structure.

3.5.2.6 medianILSFitness

```
vector<double> SearchAlgorithmResultsAnalysis::medianILSFitness
```

List of the Median Iterative Local Search fitness from each SearchAlgorithmResults structure.

3.5.2.7 medianLSFitness

```
vector<double> SearchAlgorithmResultsAnalysis::medianLSFitness
```

List of the Median Local Search fitness from each SearchAlgorithmResults structure.

3.5.2.8 processTimesBS

```
vector<double> SearchAlgorithmResultsAnalysis::processTimesBS
```

List of process times in ms for each Blind Search in SearchAlgorithmResults structure.

3.5.2.9 processTimesILS

```
vector<double> SearchAlgorithmResultsAnalysis::processTimesILS
```

List of process times in ms for each Iterative Local Search in SearchAlgorithmResults structure.

3.5.2.10 processTimesLS

```
vector<double> SearchAlgorithmResultsAnalysis::processTimesLS
```

List of process times in ms for each Local Search in SearchAlgorithmResults structure.

3.5.2.11 rangesBS

```
vector<vector<double> > SearchAlgorithmResultsAnalysis::rangesBS
```

List of ranges for each Blind Search result per SearchAlgorithmResults structure.

3.5.2.12 rangesILS

```
vector<vector<double> > SearchAlgorithmResultsAnalysis::rangesILS
```

List of ranges for each Iterative Local Search result per SearchAlgorithmResults structure.

3.5.2.13 rangesLS

```
vector<vector<double> > SearchAlgorithmResultsAnalysis::rangesLS
```

List of ranges for each Local Search result per SearchAlgorithmResults structure.

3.5.2.14 standardDeviationBS

```
vector<double> SearchAlgorithmResultsAnalysis::standardDeviationBS
```

List of standard Blind Search fitness deviations per SearchAlgorithmResults structure.

3.5.2.15 standardDeviationILS

```
vector<double> SearchAlgorithmResultsAnalysis::standardDeviationILS
```

List of standard Iterative Local Search fitness deviations per SearchAlgorithmResults structure.

3.5.2.16 standardDeviationLS

```
vector<double> SearchAlgorithmResultsAnalysis::standardDeviationLS
```

List of standard Local Search fitness deviations per SearchAlgorithmResults structure.

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

- C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/DataStructs.h

Chapter 4

File Documentation

4.1 C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/Benchmark↩ Functions.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/Project2/BenchmarkFunctions2/BenchmarkFunctions.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/Project2/BenchmarkFunctions2/Data↩ Structs.h File Reference

A library of data structures for storing BenchmarkFunction data.

```
#include <string>
#include <vector>
```

Classes

- struct FunctionData

  Function Data Function Data Structure, to keep track of all the data used for the Benchmark Functions.
- struct FunctionAnalysis

  Function Analysis Function Analysis Structure, to keep track of the analysis performed on each FunctionData structure. Basically, it compiles and holds the averages of the calculations performed for each function.
- struct SearchAlgorithmResults

  Results of Search Algorithms. Search Algorithm Results Structure, to keep track of the search algorithm output.
- struct SearchAlgorithmResultsAnalysis

  Search Algorithm Analysis Search Algorithm Analysis Structure, to keep track of the analysis performed on each SearchAlgorithmResults structure. Basically, it compiles and holds the averages of the calculations performed for each search algorithm.

4.3.1 Detailed Description

A library of data structures for storing BenchmarkFunction data.

Author

Al Timofeyev

Date

April 17, 2019

4.4 C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/Filename↩ Constants.h File Reference

A list of input and output filenames. The input files are where the matrices are stored. The output files are where the results from the benchmark functions are stored.

```
#include <string>
```

Variables

- string **out_schefelsFilename** = "Schefels.csv"
- string **out_deJongsFilename** = "DeJongs.csv"
- string **out_rosenbrockFilename** = "Rosenbrock.csv"
- string **out_rastriginFilename** = "Rastrigin.csv"
- string **out_griewangkFilename** = "Griewangk.csv"
- string **out_sEnvSWaveFilename** = "SEnvSWave.csv"
- string **out_strchVSinWaveFilename** = "StrchVSinWave.csv"
- string **out_ackleys1Filename** = "Ackleys1.csv"
- string **out_ackleys2Filename** = "Ackleys2.csv"
- string **out_eggHolderFilename** = "EggHolder.csv"
- string **out_ranaFilename** = "Rana.csv"
- string **out_pathologicalFilename** = "Pathological.csv"
- string **out_michalewiczFilename** = "Michalewicz.csv"
- string **out_mastersCosWaveFilename** = "MastersCosWave.csv"
- string **out_quarticFilename** = "Quartic.csv"
- string **out_levyFilename** = "Levy.csv"
- string **out_stepFilename** = "Step.csv"
- string **out_alpineFilename** = "Alpine.csv"

4.4.1 Detailed Description

A list of input and output filenames. The input files are where the matrices are stored. The output files are where the results from the benchmark functions are stored.

Author

Al Timofeyev

Date

March 28, 2019

4.5 C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/Process↩ Functions.cpp File Reference

```
#include "FilenameConstants.h"
#include "ProcessFunctions.h"
```

4.6 C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/Process←Functions.h File Reference

A class used to process matrices against Benchmark Functions and analyze the results.

```
#include <iostream>
#include <fstream>
#include <random>
#include <chrono>
#include "utilities.h"
#include "DataStructs.h"
#include "SearchAlgorithms.h"
```

Classes

- class ProcessFunctions

  A class used to process matrices against Benchmark Functions and analyze the results.

Macros

- #define DEFAULT_NUM_OF_DIMENSIONS 30
- #define DEFAULT_NUM_OF_VECTORS 30
- #define BOUNDARY_MIN -500.0
- #define BOUNDARY_MAX 500.0

4.6.1 Detailed Description

A class used to process matrices against Benchmark Functions and analyze the results.

Author

Al Timofeyev

Date

April 17, 2019

4.6.2 Macro Definition Documentation

4.6.2.1 BOUNDARY_MAX

```
#define BOUNDARY_MAX 500.0
```

The default maximum boundary for the elements generated.

4.6.2.2 BOUNDARY_MIN

```
#define BOUNDARY_MIN -500.0
```

The default minimum boundary for the elements generated.

4.6.2.3 DEFAULT_NUM_OF_DIMENSIONS

```
#define DEFAULT_NUM_OF_DIMENSIONS 30
```

The default minimum number of dimensions.

4.6.2.4 DEFAULT_NUM_OF_VECTORS

```
#define DEFAULT_NUM_OF_VECTORS 30
```

The default number of vectors per matrix.

4.7 C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/Search↩ Algorithms.cpp File Reference

A library of Search Algorithms.

```
#include "SearchAlgorithms.h"
```

Functions

- double blindSearch (int iterations, vector< double > argBest, double fitness0, int functionID, int rows, int columns, double minBound, double maxBound)

  Implementations of a Blind Search Algorithm.
- double localSearch (vector< double > argBest, int functionID, double alpha)

  Implementations of a Local Search Algorithm.
- vector< double > iterativeLocalSearch (int iterations, vector< double > argBest, int functionID, double alpha)

  Implementations of a Iterative Local Search Algorithm.
- vector< double > createNeighborhood (vector< double > origVect, double origFitness, int functionID, double alpha)

  Create a neighborhood of the original vector.

4.7.1 Detailed Description

A library of Search Algorithms.

Author

Al Timofeyev

Date

April 17, 2019

4.7.2 Function Documentation

4.7.2.1 blindSearch()

```
double blindSearch (
              int iterations,
              vector< double > argBest,
              double fitness0,
              int functionID,
              int rows,
              int columns,
              double minBound,
              double maxBound )
```

Implementations of a Blind Search Algorithm.

Note

This function makes a call to utilities.h --> createMatrix().
This function makes a call to utilities.h --> calculateFitnessOfMatrix().
This function makes a call to utilities.h --> quicksort().

Parameters

iterations	The number of times the Blind Search has to run.
argBest	The initial vector of doubles that produced the initial best fitness.
fitness0	The initial best fitness produced by argBest.
functionID	The ID of the function to use for calculating the fitness.
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 best fitness found using Blind Search.

4.7.2.2 createNeighborhood()

```
vector<double> createNeighborhood (
              vector< double > origVect,
              double origFitness,
              int functionID,
              double alpha )
```

Create a neighborhood of the original vector.

Note

This is used in Local Search and Iterative Local Search algorithms.

Parameters

origVect	The original vector.
origFitness	The fitness or origVect.
functionID	The ID of the function to use for calculating the fitness.
alpha	The value used to mutate the original vector.

Returns

The neighborhood of the original vector.

4.7.2.3 iterativeLocalSearch()

```
vector<double> iterativeLocalSearch (
            int iterations,
            vector< double > argBest,
            int functionID,
            double alpha )
```

Implementations of a Iterative Local Search Algorithm.

Note

This function makes a call to utilities.h --> calculateFitnessOfVector().

Parameters

iterations	The maximum number of times the Iterative Local Search can run.
argBest	The initial vector of doubles that produced the initial best fitness.
functionID	The ID of the function to use for calculating the fitness.
alpha	The value used to mutate the argBest vector.

Returns

A vector of best fitness' found using Iterative Local Search.

4.7.2.4 localSearch()

```
double localSearch (
            vector< double > argBest,
            int functionID,
            double alpha )
```

Implementations of a Local Search Algorithm.

Note

This function makes a call to utilities.h --> calculateFitnessOfVector().

Parameters

argBest	The initial vector of doubles that produced the initial best fitness.
functionID	The ID of the function to use for calculating the fitness.
alpha	The value used to mutate the argBest vector.

Returns

The best fitness found using Local Search.

4.8 C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/Search↵ Algorithms.h File Reference

A library of Search Algorithms.

```
#include <vector>
#include "utilities.h"
```

Functions

- double blindSearch (int iterations, vector< double > argBest, double fitness0, int functionID, int rows, int columns, double minBound, double maxBound)

    Implementations of a Blind Search Algorithm.
- double localSearch (vector< double > argBest, int functionID, double alpha)

    Implementations of a Local Search Algorithm.
- vector< double > iterativeLocalSearch (int iterations, vector< double > argBest, int functionID, double alpha)

    Implementations of a Iterative Local Search Algorithm.
- vector< double > createNeighborhood (vector< double > origVect, double origFitness, int functionID, double alpha)

    Create a neighborhood of the original vector.

4.8.1 Detailed Description

A library of Search Algorithms.

Author

Al Timofeyev

Date

April 17, 2019

4.8.2 Function Documentation

4.8.2.1 blindSearch()

```
double blindSearch (
             int iterations,
             vector< double > argBest,
             double fitness0,
             int functionID,
             int rows,
             int columns,
             double minBound,
             double maxBound )
```

Implementations of a Blind Search Algorithm.

Uses Blind Search algorithm and returns the best fitness found.

Note

This function makes a call to utilities.h --> createMatrix().
This function makes a call to utilities.h --> calculateFitnessOfMatrix().
This function makes a call to utilities.h --> quicksort().

Parameters

iterations	The number of times the Blind Search has to run.
argBest	The initial vector of doubles that produced the initial best fitness.
fitness0	The initial best fitness produced by argBest.
functionID	The ID of the function to use for calculating the fitness.
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 best fitness found using Blind Search.

4.8.2.2 createNeighborhood()

```
vector<double> createNeighborhood (
             vector< double > origVect,
             double origFitness,
             int functionID,
             double alpha )
```

Create a neighborhood of the original vector.

Creates a neighborhood of a vector using an alpha value and original vector.

Note

This is used in Local Search and Iterative Local Search algorithms.

Parameters

origVect	The original vector.
origFitness	The fitness or origVect.
functionID	The ID of the function to use for calculating the fitness.
alpha	The value used to mutate the original vector.

Returns

The neighborhood of the original vector.

4.8.2.3 iterativeLocalSearch()

```
vector<double> iterativeLocalSearch (
            int iterations,
            vector< double > argBest,
            int functionID,
            double alpha )
```

Implementations of a Iterative Local Search Algorithm.

Uses Iterative Local Search algorithm and returns a list of the best fitness found.

Note

This function makes a call to utilities.h --> calculateFitnessOfVector().

Parameters

iterations	The maximum number of times the Iterative Local Search can run.
argBest	The initial vector of doubles that produced the initial best fitness.
functionID	The ID of the function to use for calculating the fitness.
alpha	The value used to mutate the argBest vector.

Returns

A vector of best fitness' found using Iterative Local Search.

4.8.2.4 localSearch()

```
double localSearch (
            vector< double > argBest,
            int functionID,
            double alpha )
```

Implementations of a Local Search Algorithm.

Uses Local Search algorithm and returns the best fitness found.

Note

This function makes a call to utilities.h --> calculateFitnessOfVector().

Parameters

argBest	The initial vector of doubles that produced the initial best fitness.
functionID	The ID of the function to use for calculating the fitness.
alpha	The value used to mutate the argBest vector.

Returns

The best fitness found using Local Search.

4.9 C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/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.
- vector< vector< double > > createMatrix (int rows, int columns, double minBound, double maxBound)

  Creates a matrix of doubles using Mersenne Twister.
- 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.

4.9.1 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

4.9.2 Function Documentation

4.9.2.1 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.

4.9.2.2 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.

4.9.2.3 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.

4.9.2.4 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.

4.9.2.5 createMatrix()

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

Creates a matrix of doubles using Mersenne Twister.

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.

4.9.2.6 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.

4.9.2.7 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.

4.9.2.8 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.

4.9.2.9 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.

4.9.2.10 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

> 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).

4.9.2.11 quicksort() [2/2]

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

A normal Quicksort implementation for vector arrays of doubles.

Note

> 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).

4.9.2.12 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.

4.9.2.13 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.

4.10 C:/Users/altim/Documents/School/CS471/Project2/BenchmarkFunctions2/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.
- vector< vector< double > > createMatrix (int rows, int columns, double minBound, double maxBound)

  Creates a matrix of doubles using Mersenne Twister.
- 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.

4.10.1 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

4.10.2 Function Documentation

4.10.2.1 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.

4.10.2.2 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.

4.10.2.3 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.

4.10.2.4 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.

4.10.2.5 createMatrix()

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

Creates a matrix of doubles using Mersenne Twister.

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.

4.10.2.6 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.

4.10.2.7 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.

4.10.2.8 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.

4.10.2.9 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.

4.10.2.10 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

> 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).

4.10.2.11 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

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).

4.10.2.12 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.

4.10.2.13 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.

Index